

# Processing hidden Markov models using recurrent neural networks for biological applications

Pavan Kumar Rallabandi



UNIVERSITY of the  
WESTERN CAPE

A Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science at the Faculty of Natural Sciences, University of the Western Cape

Supervisor: Prof. Kailash C. Patidar

November 2013

# KEYWORDS

Artificial intelligence techniques

Machine learning algorithms and technologies

Neural networks and advanced models

Hidden markov model algorithms

Hybrid systems

Connectionist and symbolic learning

Automata theory and formal languages

Bioinformatic applications

Knowledge based systems frameworks.



# ABSTRACT

## **Processing hidden Markov models using recurrent neural networks for biological applications**

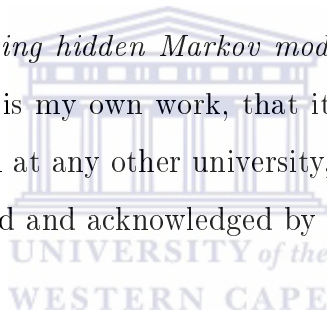
**Pavan Kumar Rallabandi**

**PhD thesis, Department of Computer Science, Faculty of Natural Sciences, University of the Western Cape.**

In this thesis, we present a novel hybrid architecture by combining the most popular sequence recognition models such as Recurrent Neural Networks (RNNs) and Hidden Markov Models (HMMs). Though sequence recognition problems could be potentially modelled through well trained HMMs, they could not provide a reasonable solution to the complicated recognition problems. In contrast, the ability of RNNs to recognize the complex sequence recognition problems is known to be exceptionally good. It should be noted that in the past, methods for applying HMMs into RNNs have been developed by other researchers. However, to the best of our knowledge, no algorithm for processing HMMs through learning has been given. Taking advantage of the structural similarities of the architectural dynamics of the RNNs and HMMs, in this work we analyze the combination of these two systems into the hybrid architecture. To this end, the main objective of this study is to improve the sequence recognition/classification performance by applying a hybrid neural/symbolic approach. In particular, trained HMMs are used as the initial symbolic domain theory and directly encoded into appropriate RNN architecture, meaning that the prior knowledge is processed through the training of RNNs. Proposed algorithm is then implemented on sample test beds and other real time biological applications.

# DECLARATION

I declare that the *Processing hidden Markov models using recurrent neural networks for biological applications* is my own work, that it has not been submitted before for any degree or examination at any other university, and that all sources I have used or quoted have been indicated and acknowledged by complete references.



Pavan Kumar Rallabandi

November 2013

Signed .....



# ACKNOWLEDGEMENT

I would like to thank my supervisor Prof. Kailash C. Patidar, the University of the Western Cape, families and friends, and other relatives. Of course without my parent's support I would not have been here. Your love and blessings got me through. Finally, I acknowledge the undivided support of my wife in this journey.



# DEDICATION

I dedicate this thesis to my

parents,



and

GOD.

# Contents

<b>Keywords</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Declaration</b>	<b>iii</b>
<b>Aknowledgement</b>	<b>iv</b>
<b>Dedication</b>	<b>v</b>
<b>List of Tables</b>	<b>viii</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Publications</b>	<b>x</b>
<b>1 General Introduction</b>	<b>1</b>
1.1 Rational and motivation for this study . . . . .	1
1.2 Preliminaries . . . . .	3
1.3 Literature review . . . . .	12
1.4 Aims and objectives of this thesis . . . . .	16
1.5 Outline of the thesis . . . . .	20
<b>2 Architectures of the proposed hybrid systems</b>	<b>22</b>
2.1 Introduction . . . . .	22

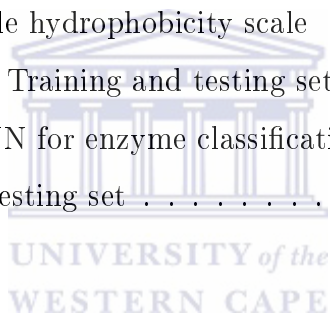


2.2	Concepts of neural networks . . . . .	25
2.3	Recurrent Neural Networks (RNNs) . . . . .	33
2.4	Hidden Markov Models (HMMs) . . . . .	44
2.5	Summary . . . . .	49
<b>3</b>	<b>Derivation and training of hybrid systems using HMM and RNN</b>	<b>51</b>
3.1	Introduction . . . . .	51
3.2	RNNs based on HMMs . . . . .	59
3.3	Real time recurrent learning for the hybrid HMM-RNN system . . . . .	65
3.4	Applications of hybrid systems . . . . .	69
3.5	Summary . . . . .	71
<b>4</b>	<b>Sample test beds and implementation results</b>	<b>72</b>
4.1	Introduction . . . . .	72
4.2	Finite-state automata and knowledge representation . . . . .	77
4.3	Experimental results . . . . .	84
4.4	Summary . . . . .	88
<b>5</b>	<b>Real time application - Enzyme classification</b>	<b>89</b>
5.1	Introduction . . . . .	89
5.2	Central design of Molecular Biology . . . . .	95
5.3	Enzyme classification . . . . .	103
5.4	Results and discussion . . . . .	107
5.5	Summary . . . . .	108
<b>6</b>	<b>Concluding remarks and scope for further research</b>	<b>109</b>
	<b>Bibliography</b>	<b>111</b>



# List of Tables

4.3.1 Hidden Markov model parameters . . . . .	87
4.3.2 Single order recurrent neural network . . . . .	87
4.3.3 Learning deterministic finite state automation using hybrid HMM-RNN	88
5.3.1 Kyte and Doolittle hydrophobicity scale . . . . .	104
5.3.2 HMM-RNN-EC - Training and testing set percentages . . . . .	105
5.4.1 Hybrid HMM-RNN for enzyme classification . . . . .	107
5.4.2 Accuracy of the testing set . . . . .	108



# List of Figures

2.2.1 Architectures of feed-forward and recurrent neural networks . . . . .	30
2.3.1 Architectures of RNN . . . . .	35
2.3.2 First order RNNs . . . . .	37
2.3.3 Second order RNNs . . . . .	38
2.3.4 NARX RNNs . . . . .	40
2.3.5 Long short term memory - RNN . . . . .	41
2.4.1 Hidden Markov model sample with emission states . . . . .	46
3.1.1 Knowledge based neural network . . . . .	56
3.2.1 Hybrid HMM-RNN Architecture . . . . .	63
4.2.1 Example of a deterministic finite state automata . . . . .	79
4.2.2 Hidden Markov model with emitting states . . . . .	82
4.3.1 Ten state deterministic finite state automata . . . . .	85
5.2.1 Example of a typical DNA molecule . . . . .	95
5.2.2 Hybrid HMM-RNN architecture . . . . .	97
5.2.3 Topology of a profile HMMs . . . . .	98
5.2.4 Tested sequence - Isomerase . . . . .	101

# List of Publications

We are preparing following manuscripts that we will be submitting soon for publication in international journals:

1. Pavan K. Rallabandi and Kailash C. Patidar, An optimization technique for hybrid systems of hidden markov models and recurrent neural networks.
2. Pavan K. Rallabandi and Kailash C. Patidar, Processing of hybrid hidden markov model and recurrent neural network on dynamical systems.
3. Pavan K. Rallabandi and Kailash C. Patidar, Enzyme classification using hybrid hidden markov model and recurrent neural network.

# Chapter 1

## General Introduction

This thesis deals with performance improvement of sequence recognition or classification systems. In this chapter, we present a general motivation for this study followed by some preliminary concepts that will be useful for this study. Then we make problem statements, list the research hypotheses, illustrate technical objectives and present research methodology. This is augmented with a thorough literature review highlighting some recent research in the field. Finally, we present the outline of the rest of the thesis.

### 1.1 Rational and motivation for this study

Sequence recognition is a major step in many applications ranging from processing (e.g., speech recognition, signature verification, time series modeling and prediction) to bioinformatics (e.g., DNA analysis). Hidden Markov models (HMMs) are one of the most popular techniques for sequence modeling and classification because they are easy to train. However, HMMs generally do not perform satisfactorily on difficult recognition problems. Recurrent neural networks (RNNs) are alternative methods for modeling sequences. One may note that, RNNs have excellent generalization performance, but training RNNs can be very difficult.

Previously, it has been shown by several researchers (see, e.g., [79, 86, 180] and some



of the references therein) that recurrent neural networks are excellent tools for processing existing domain theories with hidden states through learning and that learned knowledge can be extracted in symbolic form. Recently, there has been a lot of interest in combining symbolic and neural learning. There are different ways in which neural and symbolic learning can be combined to solve a given learning task. It is, however, difficult to interpret the knowledge stored in neural networks.

Two paradigms of computing are developed within the field of machine learning: *Symbolic* systems learning, that is used as the construction of explicit representations of knowledge, and *Connectionist* systems learning, that is used to modify the strength of connections between interconnected units. The primary motivation for studying connectionist systems is that they present an alternative to the conventional *von Neumann* model of computation which states that ‘Learning replaces a priori program development’ [140]. Within this field of machine learning, the computational advantages of such neural computing include efficient processing once in operation, the ability to learn knowledge from noisy and incomplete data and to easily adapt to new environments.

Although much research has been done to overcome the weaknesses of neural networks, many of the challenges still remain unaddressed. Some of the open problems in neuro-computing include the difficulty in interpreting the knowledge stored in neural networks and the lack of clear design rules. Besides these, there is no guarantee that a network can be trained in a finite amount of time.

Theoretical models of computation have been shown to be well-suited for investigating the representational capabilities of recurrent neural network architectures. To this end, our research is focused on learning and embedding finite-state automata in recurrent neural networks. Thus, it has not only contributed to the theoretical foundation of recurrent neural networks, but it has also connected a classical discipline of Computer Science, namely, the theory of computation, with an emerging area of artificial intelligence. We are particularly interested in how the nonlinear dynamical behavior of recurrent neural networks can be exploited for computation, and what impact limitations and extensions of the basic network architectures have on their representational

and computational capabilities.

We also investigate several questions of learning, i.e., how can learning be made more efficient by pre-structuring networks with prior knowledge or how can we guide learning such that a network can find a solution to a learning problem? We also investigate methods for building hybrid systems, i.e., systems which combine the strengths of various paradigms while avoiding their respective weaknesses. One such example is the training or processing of hidden Markov models with recurrent neural networks.

Before we proceed further, let us discuss some necessary concepts that are required for this study.

## 1.2 Preliminaries

In this section, we present some of the key concepts and related information that can be useful for better understanding of the rest of the thesis. In particular, we will discuss on knowledge based neuro-computing, symbolic knowledge and connectionist learning, importance of prior knowledge, significance of knowledge extraction, knowledge refinement, neural networks, learning algorithms, hidden Markov models, hybrid systems, etc.

### **Knowledge based neuro-computing:**

There has been an increasing interest in hybrid systems as more applications using hybrid models emerged in the past. Examples of such hybrid systems include combining artificial neural networks and fuzzy systems, and the use of neural networks for knowledge refinement with the use of initial domain theories. In view of this, the main objective of this research is to study the capability of recurrent neural networks to represent hybrid structures. For these structures, the representational properties of artificial intelligence and machine learning structures as well as their proofs are important for a number of reasons.

Many users of a hybrid model would like to know what can they theoretically achieve using this model? Some are interested to know the performance and capabilities of such a model whereas others need this for its justification and acceptance. In practical terms, the study of the representational capabilities includes the development of algorithms for mapping knowledge into neural networks and for extracting such knowledge from trained networks. One can also apply these techniques to the prediction of seismic events in mines where hybrid systems may not only improve prediction accuracy and reliability (i.e., high probability of prediction and low false alarm rate of large seismic events). However, the extraction of information from trained neural networks may also add some insight into the seismology of rock formations.

There are essentially two approaches for building problem-specific knowledge in an automated system: method based on hand-built classifier systems and empirical learning [181]. Hand-built systems such as expert systems are non-learning. On the other hand, the empirical learning systems are generalization through induction from specific examples. Many models and techniques within this paradigm, such as, neural networks, genetic algorithms, neuro-fuzzy logic and hidden Markov models have been studied extensively. However, to the best of our knowledge, no single technique could fully address all the problems and related issues of machine learning. Hybrid systems on the other hand, incorporate different machine learning techniques and are proved to be an effective paradigm of machine learning for many real-world applications (e.g., [67, 173, 180]). Motivated by this fact, in this thesis, we will be looking at hybrid systems for integrating connectionist and symbolic learning, known as knowledge-based neural networks.

In knowledge-based neuro-computing the emphasis is on use and representation of knowledge about an application. Explicit modeling of the knowledge represented by such a system remains a major research topic. The reason is that humans find it difficult to interpret the numeric representation of a neural network.

The key assumption of knowledge-based neuro-computing is that the knowledge is obtainable from, or can be represented by, a neuro-computing system in a form that

humans can understand, i.e., the knowledge embedded in the neuro-computing system can also be represented in a symbolic or well-structured form, such as boolean functions, automata, rules, or other familiar ways. The focus of knowledge-based computing is on methods to encode prior knowledge and to extract, refine, and revise knowledge within a neuro-computing system.

### **Symbolic knowledge and connectionist learning:**

One of the main reasons for combining connectionist learning with symbolic systems is the potential of developing intelligent systems that neither type of model alone can easily deal with. One particular flaw in neural networks is their lack of transparency. Still, neural networks have been applied very successfully in symbolic-connectionist hybrid applications to areas such as image classification [115], natural language processing [113] and financial data modeling [114].

The hybrid model facilitates the conversion from symbolic domain theory to connectionist form and vice-versa. To do this, initial domain theory is used to initialize a neural network. The knowledge is refined by neural learning on training data and symbolic knowledge is extracted from the trained network. The symbolic knowledge is revised and used to guide the adaptation of the network architecture in the symbolic component of the model. This enables an iterative process of knowledge insertion, extraction and refinement on symbolic interpretations of the knowledge stored in the neural network.

Often neural networks have shown to outperform other empirical learning methods in terms of classification accuracy, reasoning with noisy or partial data and generalization with small training sets [129, 181]. They have at least comparable accuracies to symbolic methods such as decision tree induction [6]. Towell *et al.* [180], for example, have shown that a knowledge-based neural network can outperform a standard back-propagation network as well as other related symbolic and numeric algorithms. Another compelling reason for the use of neural networks is the existence of architectures that

can model tasks of a sequential nature [192].

It should be noted that good empirical results have been achieved using the framework that combines neural and symbolic learning described above. However, the merits underlying the symbolic or connectionist approach are not yet well understood. Gaining that insight remains an important open research problem.

### **Hybrid systems:**

Extensive research has been conducted on hybrid systems and they have been applied to a diverse and growing range of problems. The combination of the two paradigms (symbolic and connectionist learning systems) has led to the development of a new, more powerful and flexible paradigm of learning called hybrid systems. These symbolic and connectionist learning systems have been supplementing each other. Connectionist structures are now recognized as complementary sub-symbolic abstractions of symbolic expressions of knowledge. Knowledge insertion, refining and extraction provides a difficult but essential link between symbolic and connectionist knowledge representation.

### **The importance of prior knowledge:**

The utility of knowledge in inductive learning has been evaluated both theoretically and empirically in [140]. It has been shown that some prior structure is required for meaningful learning [72, 127]. Many machine-learning problems beneficially utilize initial domain theory in the form of a prior knowledge. In feed-forward networks, for example, the use of prior knowledge has been analyzed in terms of generalization ability and the required sample size for valid generalization [1, 68]. Some researchers have indicated that neural network training with prior knowledge requires fewer samples for valid generalization compared to training without prior knowledge.

Fidelity of the mapping of the prior knowledge into a network is very important since a network may not be able to take full advantage of poorly encoded prior knowledge or, if the encoding alters the essence of the prior knowledge, the prior knowledge may

actually hinder the learning process.

Prior knowledge is usually represented in the form of explicit rules in symbolic form. The common form for feedforward networks is propositional logic expressions while finite-state automata are predominantly used for recurrent architectures. A number of knowledge encoding algorithms have been proposed (e.g., [102, 124, 180]). Prior knowledge can be obtained by analysis of the available domain knowledge and can be used to structure the network prior to learning [1]. Prior knowledge in the form of rules have been used to define the architecture of the initial neural network [180]. The inclusion of good quality prior knowledge in a hybrid system can help realize a number of beneficial features:

- Faster convergence: A neural network initialized with knowledge about a learning task prior to training is biased to learn the desired hypothesis more efficiently [167]. The speed up can occur in two ways: (1) The time complexity of training is decreased, and (2) fewer training examples are required to achieve acceptable results [179].
- The capability to learn incrementally: Initial domain theory can be augmented with knowledge acquired from new data presented to a hybrid system. This process is often viewed as knowledge refinement [68, 124, 180].
- The ability to configure the architecture: The use of prior knowledge obviates the need to guess an initial architecture or at the least, can restrict the combinatorial search for a good starting point or hypothesis space to explore.

Feed-forward architectures have been shown to deal successfully with applications where prior knowledge is bounded such as classification tasks [67, 181]. Recurrent networks, on the other hand, provide a means for dealing with unbounded structures [66]. Pollack [144], for example, uses recursive auto-associative memories and other researchers employ stack structures to handle unbounded knowledge such as context-free grammars [47, 131]. A number of algorithms exist for inserting prior knowledge into feedforward [67, 167] and recurrent networks [35, 66].

**Significance of knowledge extraction:**

Knowledge extraction provides a direct way of converting the knowledge stored in a neural network to a symbolic form [5, 6]. Rule extraction from neural networks is distinct from rule induction on a data set by symbolic methods. Knowledge extraction from neural networks is useful for a number of things, for example, neural network transparency, neural network validation, knowledge acquisition and discovery, knowledge transfer and refinement, etc. Andrews *et al.* [6] discussed the importance of knowledge extraction. They further explored several knowledge extraction techniques applicable to feed-forward networks.

**Knowledge refinement:**

Knowledge refinement or revision is one of the main goals of learning in hybrid systems [167]. Knowledge refinement in the connectionist component of a knowledge-based system equates to neural network learning. Approaches to guide refinement more effectively include the use of different activation functions [67] and constraining weight changes to maintain the symbolic interpretation of the network [124].

Utgoff's perceptron tree construction algorithm [185] and the knowledge based artificial neural network method of Shavlik *et al.* [180] are examples of symbolic inductive learning performed in conjunction with feed-forward neural learning. One more example of domain knowledge refinement is RuleNet [124]. Some of the benefits of knowledge refinement are

- Increased training speed,
- Better understanding of the internal operation of the network,
- Improved generalization, and
- A means of bridging the gap between symbolic and sub-symbolic levels of knowledge representation.

**Neural networks:**

Depending on the flow of signals, neural networks can be classified into two types of topologies:

- (i) *feed-forward* networks that contain only open-loop interconnections, and
- (ii) *recurrent* networks that contain one or more closed feedback paths in addition to open-loop paths.

In contrast to feed-forward networks, recurrent networks are dynamical systems whose output depends on the present state of the units in the network; learning in feedback networks corresponds to function approximation [96]. Examples of recurrent networks are the *Hopfield net* [94] with symmetric fully connected feedback neurons with discrete states and hard limiting activation function, and the *Boltzmann machine* [89] which is a stochastic version of the discrete-time Hopfield net with transfer function  $f_T$ .

**Recurrent neural networks:**

Recurrent networks are computational systems with context-varying responses, i.e., they have the ability to model two types of time-dependent behavior: The first deals with gradually settling into a solution for a complex set of conflicting constraints such as pattern completion, whereas the second concerns the modeling of pattern sequences. Recurrent networks have been successfully applied to problems ranging from formal grammars, speech and image recognition to time series prediction [58, 193].

Recurrent networks generally use some delayed output to calculate the current activation. Some represent this delayed copy explicitly by means of context units, e.g., Jordan and Elman networks, whereas others employ an implicit signal delay mechanism [107]. Examples of the latter are Hopfield networks, Boltzmann machines and second-order networks.

Recurrent network behaviour has been shown to correspond to discrete state transitions and can hence emulate finite-state automata. Although RNNs can learn stable



encodings for short-term dependencies, they are not as effective in learning long temporal relationships. Several approaches have been proposed to overcome this difficulty. These methods however, impose various constraints on the RNN architecture and the state space or learning algorithm that is employed, e.g., compressing regular sequences and discretized state spaces. Some of these approaches may be advantageous under certain circumstances. However, an approach that imposes no such limitations is more desirable in practice.

### **Learning algorithms:**

Backpropagation is the most widely applied learning algorithm for both feed-forward and recurrent neural networks. Recurrent networks can exhibit a wide range of asymptotic behaviors. The simplest is that the system reaches a stable fixed point; more complex asymptotic behaviors include limit cycles and chaos. Empirical evidence suggests that first-order networks generalize better than second-order networks of similar number of neurons [80].

There are basically two learning algorithms which can be used for training the recurrent neural network based on the back-propagation technique. A recurrent network can be unfolded in time, into a feed-forward network by adding a layer for each time step. Such a network has identical behavior to a recurrent network for a finite number of time steps and is referred as a deep multilayer network. The learning algorithm is known as back-propagation through time [159]. Real-time recurrent learning is a real time learning algorithm which updates the weights at the end of each sample string presentation with a gradient descent weight update rule. The algorithm computes the derivatives of states and outputs with respect to all weights as the network processes the sequence during the forward step. There is no unfolding performed or necessary for real time recurrent learning.

## Training methods:

The back-propagation algorithm is predominantly employed for recurrent network learning. Some variations of this algorithm are of the following forms

- Back-propagation through time which limits the upper bound time complexity to  $\Theta(n^2d)$ , where  $d$  is the depth of error back-propagation.
- Real-time recurrent learning, which computes the gradient by storing additional information about the interactions between processing units. While the memory requirements remain constant, the time complexity is  $\Theta(n^3)$ .
- The Long Short-term algorithm proposed by Schmidhuber *et al.* [92] is used to overcome the problem of long-term dependencies.
- An earlier approach by the same author that learns to ‘divide and conquer’ by recursively decomposing sequences [162]. The algorithm first constructs a multi-level hierarchy of recurrent networks and then attempts to collapse this into a single structure.
- Teacher forcing where desired output values are fed back to train previous layers instead of the output activation values given that the outputs are known for every point in the sequence.

## Hidden Markov models:

As mentioned in [150], Hidden Markov model (HMM) describes a process, which goes through a finite number of states whilst generating a signal of either discrete or continuous nature. The model probabilistically links the observed signal to the state transitions in the system. A HMM is parameterized through a matrix of transition probabilities between states and output probability distributions for observed signal frames given the internal process state. These probabilities are used in the algorithms that are used for achieving the desired results. A typical way to combine HMMs and neural networks

is to replace the Gaussian density function estimates of emission probabilities by neural networks.

### 1.3 Literature review

Below we present a critical overview of the relevant literature on the topic in a chronological order. More specific works are reviewed in the individual chapters.

In the first part of this review, we discuss some works on neural networks and optimization techniques.

Fuzzy grammatical inference comparison using the genetic algorithm and the real time recurrent learning algorithm was given by Blanco *et al.* [25]. The utilization of Recurrent Neural Networks is not as impressive as Feed-forward Neural Networks. Training algorithms for Recurrent Neural Networks, based on the error gradient, are very unstable in their search for a minimum and require much computational time when the number of neurons is high. These authors presented a real coded genetic algorithm that uses the appropriate operators for this encoding type to train Recurrent Neural Networks.

A method for sequential supervised learning that exploits explicit knowledge of short and long-range dependencies was given by Alessiso *et al.* [36]. They designed an architecture consists of a recursive and bi-directional neural network that takes as input a sequence along with an associated interaction graph. The interaction graph models (partial) knowledge about long-range dependency relations. They tested the method on the prediction of protein secondary structure, a task in which relations due to beta-strand pairings and other spatial proximities are known to have a significant effect on the prediction accuracy. In this particular task, interactions can be derived from knowledge of protein contact maps at the residue level. Their validated and research results show that prediction accuracy can be significantly boosted by the integration of interaction graphs.

A learning method for some weak and strong convergence results is presented by Wu

*et al.* [196] indicating that the gradient of the error function goes to zero and the weight sequence goes to a fixed point, respectively. An online gradient learning method for back-propagation neural networks with a single hidden layer is been considered. The conditions on the activation function and the learning rate to guarantee the convergence are relaxed compared with the existing results.

Zhang and Cao [205], designed a method for analyzing the convergence performance of neural networks ranking algorithm by means of the given samples and approximation property of neural networks. Their research results showed the upper bounds of convergence rate can be considerably tight and independent of the dimension of input space when the target function satisfies some smooth condition. Their final results imply that neural networks are able to adapt to ranking function in the instance space.

In [195], Wu and Zeng developed a general method or class of memristor-based recurrent neural networks with time-varying delays. Conditions on the nondivergence and global attractivity are established by using local inhibition, respectively. They even studied the exponential convergence of the networks using local invariant sets. They analyzed results from the theory of differential equations with discontinuous right-hand sides.

A penalty based recurrent neural network method for solving a class of constrained optimization problems with generalized convex objective functions was given by Alireza *et al.* [97]. Their model has a simple structure described by using a differential inclusion and also applicable for any non-smooth optimization problem with affine equality and convex inequality constraints, provided that the objective function is regular and pseudo-convex on feasible region of the problem. Their validation results indicated that the state vector of the proposed neural network globally converges to and stays thereafter in the feasible region in finite time, and converges to the optimal solution set of the problem.

In [31], Buse and Mutlu explained a new model for radial basis function based classification neural network named as generalized classifier neural network. Unlike other radial basis function based neural networks such as generalized regression neural

network and probabilistic neural network, their generalized classifier neural network has five layers. They are input, pattern, summation, normalization and output layers. In addition to topological difference, the neural network proposed in this paper has gradient descent based optimization of smoothing parameter approach and diverge effect term added calculation improvements. Diverge effect term is an improvement on summation layer calculation to supply additional separation ability and flexibility. They compared the performance of generalized classifier neural network with that of the probabilistic neural network, multilayer perceptron algorithm and radial basis function neural network on 9 different data sets, and with that of a generalized regression neural network on 3 different data sets.

Now we discuss some research works on hybrid HMMs-RNNs and their biological applications.

A new learning architecture for sequences analyzed on short-term basis was given by Diego *et al.* [53], but not assuming stationarity within each frame. Hidden Markov models have been found very useful for a wide range of applications in machine learning and pattern recognition. The training algorithms for all the parameters in the composite model are developed using the expectation-maximization framework. They worked on two experiments with artificial and real data: model-based denoising and speech recognition. They found that the defined model and learning algorithm are more effective than previous approaches based on isolated hidden Markov trees. In the case of the Doppler benchmark sequence, with 1024 samples and additive white noise, their method reduced the mean squared error from 1.0 to 0.0842.

Alignment-free classifiers presented by Guillermin *et al.* [82] are especially useful in the functional classification of protein classes with variable homology and different domain structures. They developed three non-linear models for RNase III class prediction: Decision Tree Model (DTM), Artificial Neural Networks (ANN) model and Hidden Markov Model (HMM). The first two are alignment-free approaches, using TIs as input predictors. Their performances were compared with a non-classical HMM, modified according to our amino acid clustering strategy. The alignment-free models

showed similar performances on the training and the test sets reaching values above 90% in the overall classification. The non-classical HMM showed the highest rate in the classification with values above 95% in training and 100% in test. They mentioned that as compared to the higher accuracy of the HMM, the DTM showed simplicity for the RNase III classification with low computational cost. They evaluated such simplicity with respect to HMM and ANN models for the functional annotation of a new bacterial RNase III class member, isolated and annotated by their group.

A reliable model miRANN given by Eamin *et al.* [56] is a supervised machine learning approach. MicroRNA (miRNA) is a special class of short noncoding RNA that serves pivotal function of regulating gene expression. The computational prediction of new miRNA candidates involves various methods such as learning methods and methods using expression data. MiRANN used known pre-miRNAs as positive set and a novel negative set from human CDS regions. The number of known miRNAs is now huge and diversified that could cover almost all characteristics of unknown miRNAs which increases the quality of the result (99.9% accuracy, 99.8% sensitivity, 100% specificity) and provides a more reliable prediction. They mentioned that MiRANN performs better than other state-of-the-art approaches and declares to be the most potential tool to predict novel miRNAs. They validated and tested the result using a previous negative set.

A new model presented by Sepideh *et al.* [164] investigated the bidirectional RNNs to make the strong correlations between secondary structure elements, types of modular reciprocal recurrent neural networks (MRR-NN) are examined. They indicated that precise prediction of protein secondary structures from the associated amino acids sequence is of great importance in bioinformatics and yet a challenging task for machine learning algorithms.

A SARS genome given by Weichen *et al.* [38] demonstrated, how to conduct training and derive the corresponding results. These authors reported the discovery of strong correlations between protein coding regions and the prediction errors when using the simple recurrent network to segment genome sequences. They mentioned that the

distribution of prediction error indicates how the underlying hidden regularity of the genome sequences and the results are consistent with the finding of biologists: predicated protein coding features of SARS genome. This implies that the simple recurrent network is capable of providing new features for further biological studies when applied on genome studies. The HA gene of influenza A subtype H1N1 is also analyzed in a similar way.

Now before we move onto the next section dealing with the aims and objectives in this thesis, we would like to mention that some other relevant works are appropriately reviewed in the individual chapters.

## 1.4 Aims and objectives of this thesis

In this section, we present the main aims and objectives of this study with an insights to the sequence recognition or classification performance systems. Often, first order HMMs are deployed in practice. This means that state transition probabilities are dependent only on the previous state. This assumption is unrealistic for many real world applications of HMMs. Researchers have shown that RNNs can learn higher order dependencies from training data. Furthermore, the number of states in HMM needs to be fixed beforehand. Few other research work indicated how ANNs can be pruned (e.g., optimal brain damage) or extended during training to achieve higher discriminative and generalization performance. The structural similarity between the alpha calculation procedure in hidden Markov models and recurrent neural networks is the foundation for the mapping HMM  $\rightarrow$  RNN. To begin with, let us consider the equation of the forward algorithm of HMM for the calculation of  $P(O/\lambda)$ :

$$\alpha_j^t = \left( \sum_i^N \alpha_i^{t-1} a_{ij} \right) \cdot b_j(o^t),$$

where  $\alpha_i$ 's are the initial states,  $a_{ij}$ 's are the transition probabilities, and  $b_j(o^t)$  are observation probabilities at time  $t$ . The above alpha calculation is inherently recurrent

and bares resemblance to the recursion in a vanilla RNN:

$$x_i^t = f \left( \sum_j^N x_j^{t-1} w_{ji} \right), \quad (1.4.1)$$

where  $x_j$ s are input units and  $w_{ji}$ 's are the weights on the network.

### **Aims:**

Recurrent neural networks can generate outputs based on a given input and state that is characteristic of such dynamical behavior. Also, recurrent neural networks have the capability to learn finite-state behavior and finite-state representations can be extracted from such networks. Furthermore, it is known that neural networks can be modified constructively or destructively using criteria based on the number of network parameters as adaptation measure. With this view, this thesis investigates the following open problem:

The aim of this project is to improve the sequence recognition performance by applying a hybrid neural or symbolic approach. Trained HMMs are used as an initial symbolic domain theory, which can be directly encoded into appropriate recurrent neural network architecture. Training the recurrent neural network refines this prior knowledge. A symbolic representation of the refined HMM is extracted from the trained network if appropriate. We therefore develop a gradient descent algorithm for processing the knowledge stored in trained HMMs in this study.

The big hypothesis is that these HMM-RNN can improve classification performance through HMMs. This hypothesis can then be subdivided into following:

- HMM-RNNs cannot only represent HMMs, but also learn through training (e.g., gradient descent learning algorithm) the dynamics of the underlying unknown dynamical process.
- A HMM-RNN initialized with a HMM can then refine the HMM.



- Since the HMM-RNN contains all the information about the model of the dynamical process, it ought to be possible to extract a symbolic representation, e.g., in the form of a HMM.
- HMM-RNN can learn higher-order dependencies. This may tie in together with big hypothesis, but can be investigated independently.

### Technical Objectives:

In order to validate the above research hypotheses and to achieve the set goals we have to address the following technical objectives:

- The development of gradient descent algorithm for training HMM-RNN architecture.
- The insertion of domain knowledge and demonstrate faster RNN training on toy problems.
- Testing of HMM-RNN architecture initialized with prior knowledge will be able to refine HMM.
- The construction of a hybrid system of symbolic rules and recurrent networks for enhanced domain theory acquisition and refinement.
- Extractions of refined HMMs from trained RNNs.
- To explore the usefulness of such a system as applied to synthetic and real-world predictive learning problems.

### Methodology:

Here we present the entire methodology in order to achieve the above mentioned aims and objectives of this thesis and finally the total structure of the thesis has been explained.

A neural-symbolic hybrid system implements a framework of learning that

1. initializes a network with a priori domain knowledge,
2. refines this knowledge by training the network, and
3. extracts a refined domain theory from the trained network.

Our method of research proceeds from the investigation and development of a training algorithm that meets the first technical objective. Once the performance of the algorithm has been established, we apply domain/prior knowledge in the new architecture and demonstrate faster recurrent neural network training on sample problems then we use to test it on real world problems. After we perform good training results then we will do extraction of refined HMMs from trained RNNs and finally we apply to real world problems.

Network adaptable weights are initialized with random values drawn according to some distribution. Using numerical optimization methods (e.g., gradient descent techniques, simulated annealing), the network is trained on some known data to perform a certain task (e.g., pattern classification) until some training criterion is met. After successful training, a network can take advantage of its generalization capabilities to perform the intended task on arbitrary data. Notice that during the entire process, the knowledge remains hidden in a network adaptable connections, hence the name connectionist representation.

The above training paradigm can be enriched with symbolic knowledge in the following way (symbolic representation): Prior knowledge about a task (initial domain knowledge) is used to initialize a network prior to training. This requires a translation of the information from a symbolic into a connectionist representation. The particular method for converting the symbolic into a representation of knowledge into its equivalent connectionist representation depends on the kind of symbolic knowledge, the learning task, and the network model used for learning.

To date, most efforts are directed towards encoding prior knowledge by programming some network weights to specified values instead of choosing small random values.

The programmed weights define a starting point for the research of a solution in weight space. The premise is that a better solution will be found faster compared to starting the search from a random point in weight space. Examples of this approach include pre-structuring of feed-forward networks with boolean concepts and imposing rotation invariance in neural networks for image recognition. The choice of network architecture itself represents an implicit use of prior knowledge about an application.

Once a network has succeeded in learning a task as measured by its performance on the training data, it may be useful to extract the learned knowledge. The question arises whether it is possible to extract an adequate symbolic representation of the knowledge learned by a network, i.e., a representation that captures the essence of the learned knowledge. In many cases, the extracted knowledge may only approximate a network's true knowledge; however, it is also possible for the extracted symbolic representation to exceed the accuracy of the knowledge stored in a trained network.

Finally, we compare the performance of the HMM approach with that of our hybrid approach using real time applications such as enzyme classification, etc. Then we attempt to determine the order of the best HMM by augmenting the network training procedure with the capability to prune the weights connecting the input with the hidden state neurons. Finally, we study the feasibility of extracting a refined version of the HMM in symbolic form.

## 1.5 Outline of the thesis

The remainder of the thesis is organized as follows.

In Chapter 2, we discuss the fundamentals of neural network architectures, RNNs, HMMs, and the hybrid systems which are the combinations of RNNs and HMMs.

In Chapter 3, we discuss the hybrid systems which combine the strengths of intelligent system paradigms such as neural networks, expert systems, fuzzy logic and the combination of RNNs and HMMs and the hybrid architecture of the currently developed system and gradient descent algorithm using real time recurrent learning of

RNN.

In Chapter 4, we discuss the implementation of the gradient descent algorithm using real time recurrent learning (RTRL) of RNN on sample test bed in the form of finite automata theory which focuses on learning and training the deterministic finite state automata. We also present the results obtained by using this approach.

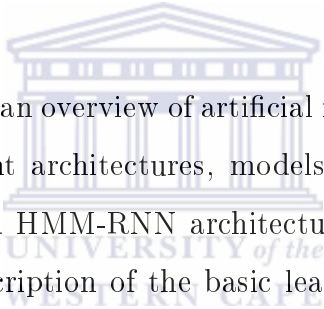
In Chapter 5, we discuss the implementation of hybrid HMM-RNN for some real world problems. As an example, we discuss its application for enzyme classification.

In Chapter 6, we present some concluding remarks where we also present scope for further research.



# Chapter 2

## Architectures of the proposed hybrid systems



In this chapter, we present an overview of artificial intelligence; machine learning; other related concepts. Relevant architectures, models and methods useful for designing and developing the hybrid HMM-RNN architecture are discussed extensively. Then we provide a detailed description of the basic learning algorithms, methods and the architectures of neural networks, first order and second order recurrent neural networks, different types of networks.

### 2.1 Introduction

The term “Artificial Intelligence” (AI) is defined as the simulation of human intelligence on a machine, so as to make the machine efficient to identify and use the right place of “Knowledge” at a given step of solving a problem [93]. A system capable of planning and executing the right task at the right time is generally called rational. Thus, AI alternatively may be stated as a subject dealing with computational models that can think and act rationally.

The subject of artificial intelligence deals with the various kinds of knowledge representation schemes, different techniques of intelligent search, various methods for re-

solving uncertainty of data and knowledge [6]. Moreover, AI has several real world applications, some of which we will discuss in this thesis.

Learning is an inherent characteristic of the human beings. By virtue of this, people, while executing similar tasks, acquire the ability to improve their performance. Here, we will provide an overview of the principle of learning that can be adhered to machines to improve their performance. Such learning is usually referred to as “Machine Learning”. Machine learning can be broadly classified into three categories:

- (i) Supervised learning,
- (ii) Unsupervised learning, and
- (iii) Reinforcement learning.

Supervised learning requires a trainer, who supplies the input-output training instances. The learning system adapts its parameters by some algorithms to generate the desired output patterns from a given input pattern. In absence of trainers, the desired output for a given input instance is not known, and consequently the learner has to adapt its parameters autonomously. Such type of learning is termed as “unsupervised learning”. The third type, the reinforcement learning, bridges a gap between supervised and unsupervised categories. In reinforcement learning, the learner does not explicitly know the input-output instances, but it receives some form of feedback from its environment. The feedback signals help the learner to decide whether its action on the environment is rewarding or punishable. The learner thus adapts its parameters based on the states (rewarding / punishable) of its actions.

There has been a great amount of work on optimization and learning algorithms applied to neural networks.

In [175], Takehiko explained the theoretical analysis of two essential training schemes for gradient descent learning in neural networks: batch and on-line training. The convergence properties of the two schemes applied to quadratic loss functions are analytically investigated. They quantified the convergence of each training scheme to the

optimal weight using the absolute value of the expected difference and the expected squared difference between the optimal weight and the weight computed by the scheme. Although on-line training has several advantages over batch training with respect to the first measure, it does not converge to the optimal weight with respect to the second measure if the variance of the per-instance gradient remains constant. However, if the variance decays exponentially, then on-line training converges to the optimal weight with respect to second measure. The final analysis or results reveals the exact degrees to which the training set size, the variance of the per-instance gradient, and the learning rate affect the rate of convergence for each scheme.

Ilya and Hinton [100] explained a method for training RNNs to predict sequences exhibits significant long-term dependencies, focusing on a serial recall task where the RNN needs to remember a sequence of characters for a large number of steps before reconstructing it. They introduced the Temporal-Kernel Recurrent Neural Network (TKRNN), which is a variant of the RNN that can cope with long-term dependencies much more easily than a standard RNN, and show that the TKRNN develops short-term memory that successfully solves the serial recall task by representing the input string with a stable state of its hidden units.

A method for comparing the maze learning performance of three artificial neural network architectures are elman recurrent neural network, a long short-term memory (LSTM) network and Mona was given by Thomas [145]. The mazes are networks of distinctly marked rooms randomly interconnected by doors that open probabilistically. The mazes are used to examine two important problems related to artificial neural networks: the retention of long-term state information and the modular use of learned information. They examined the effect of modular and non-modular training. In modular training, the door associations are trained in separate trials from the intervening maze paths and only presented together in testing trials. Their validation results indicated that all networks performed well on mazes without the context learning requirement. The Mona and LSTM networks performed well on context learning with non-modular training; the Elman performance degraded as the task length increased.

Mona also performed well for modular training; both the LSTM and Elman networks performed poorly with modular training.

The second order statistical learning framework given by Su *et al.* [78] is to study the general class of nonlinear adaptive filters with feedback realized as recurrent neural networks (RNNs). For rigour, both the so-called proper and improper-second order statistics of complex processes is taken into account and the proposed augmented complex real-time recurrent learning (ACRTRL) algorithm for RNNs has been shown to be suitable for processing a wide range of both benchmark and real-world complex processes.

In view of the above mentioned works, we present in this chapter a thorough conceptual framework that is useful for study as described in later parts of this thesis.

Rest of the chapter is organized as follows. In Section 2.2, we discuss some relevant concepts of minimization and learning algorithms. Basic elements and architectures of the neural system has been explained in Sections 2.3 and 2.4. Finally, we provide a brief summary of the work contained in this chapter in Section 2.5.

## 2.2 Concepts of neural networks

In this section, we discuss the concepts and architectures of optimization and learning algorithms, neural networks, recurrent neural networks.

Artificial neural networks are loosely modeled after biological neural systems. They learn by training from past experience data and make generalization on unseen data. Neural networks consist of artificial neurons and weights between them. The artificial neurons (processing units) transport incoming information on their outgoing connections to other units.

There are three main components when creating a functional model of the biological neuron. First one is the synapses of the neuron are modeled as weights. The strength of the connection between an input and a neuron is noted by the value of the weight. Negative weight values reflect inhibitory connections, while positive values designate



excitatory connections. The next two components model the actual activity within the neuron cell. An adder sums up all the inputs modified by their respective weights. This activity is referred to as linear combination. Finally, an activation function controls the amplitude of the output of the neuron. An acceptable range of output is usually between 0 and 1, or -1 and 1.

Mathematically, the above process can be described as

$$Y_k = \left( \sum_{j=1}^p W_{kj} X_j \right),$$

where  $Y_k$  describes the output associated with unit  $k$ ,  $W_{kj}$  are weights from unit  $k$  to unit  $j$  and  $X_j$  are the input neurons associated with the unit  $j$ .

Neural networks have been applied to many real world problems such as speech recognition [20], bio-conservation [48], gesture recognition [120], medical diagnostics [138]. Neural networks learn by training on past experience using an algorithm which modifies the interconnection weights as directed by a learning objective for a particular application. A neuron is a single processing unit which computes the weighted sum of its inputs. The output of the network relies on cooperation of the individual neurons. The learnt knowledge is distributed over a set of trained networks weights.

In general, the neural networks are characterized [96, 147] into feed-forward and recurrent neural networks. Feed-forward networks are used in application where the data does not contain time variant information while recurrent neural networks model time series sequences and possesses dynamical characteristics. Neural networks are capable of performing tasks that include pattern classification, function approximation, prediction or forecasting, clustering or categorization, time series prediction, optimization, and control.

Artificial neural networks (ANNs) with cycles are referred to as feedback, recursive, or recurrent neural networks [172]. These ANNs without cycles are referred to as feed-forward neural networks (FNNs). Well known examples of FNNs include perceptrons, radial basis function networks, Kohonen Self Organizing Maps and Hopfield nets. The

most widely used form of FNN is the multi-layer perceptron MLP [160, 197]. The units in a multi-layer perceptron are arranged in layers, with connections feeding forward from one layer to the next. Input patterns are presented to the input layer, and the resulting unit activations are propagated through the hidden layers to the output layer. This process is known as the forward pass of the network. The units in the hidden layers have (typically nonlinear) activation functions that transform the summed activation arriving at the unit. Since the output of an MLP depends only on the current input, and not on any past or future inputs, MLPs are more suitable for pattern classification. An MLP can be thought of as a function that maps from input to output vectors. Since the behaviour of the function is parameterised by the connection weights, a single MLP is capable of instantiating many different linear as well as nonlinear functions.

Neural networks, sometimes referred to as connectionist models, are parallel distributed models that constituted by several components including a set of processing units, their activation states, propagation rules, activation function, external inputs, learning algorithms, optimization parameters, etc.

Within the neural systems there are three types of units: Input units, which receive data from outside of the network; Output units, which send data out of the network, and Hidden units, whose input and output signals remain within the network. Through synaptic connections from other units, each of the non-input units in a neural network combine values that are fed into them and then they produce a single value called net input. The total input to a particular unit is simply the weighted sum of the separate outputs from the connected units plus a threshold or bias term.

Mathematically, the above can be expressed as

$$a_j = \sum_{i=1}^n w_{ji}x_i + \theta_j,$$

where  $\theta_j$  represents a bias term associated with the unit  $j$ . In the above,  $a_j$  represents the total input corresponding to unit  $j$ ,  $w_{ji}$  are the synaptic weights from unit  $j$  to unit  $i$ ,  $x_i$  is the input signal. In some cases, more complex rules for combining inputs,

for example, the  $\sigma - \pi$  propagation rules ([4])

$$a_j = \sum_{i=1}^n w_{ji} \prod_{k=1}^m x_{ik} + \theta_j.$$

are used. Here  $x_{ik}$  is the input signal from multiple nodes. Most units in neural network transform their net inputs by using a scalar-to-scalar function, called an activation function [63]. One of the popularly used activation function is the *binary step function* (also known as threshold function or Heaviside function). The output of this binary step function is limited to one of the two values, either in a form of a *sigmoid function*

$$g(x) = \frac{1}{1 + e^{-x}},$$

or in the form of a *bipolar sigmoid function*

$$g(x) = \frac{1 - e^{-x}}{1 + e^{-x}}.$$

Note that the sigmoid function especially advantageous for use in neural networks trained by back-propagation, because it is easy to differentiate, and thus can dramatically reduce the computation burden for training. It applies to applications whose desired output values are between 0 and 1. One the other hand, the bipolar sigmoid function has similar properties as those of the sigmoid function but it works well for applications that yield output values in the range of [-1,1].

Note that it is the non-linearity that makes multi-layer networks very powerful. Such non-linearity is introduced into the network using activation functions. The simple reason is that the composition of linear functions is again a linear function. Almost any nonlinear function does the job, although for back-propagation learning it must be differentiable and it helps if the function is bounded. However, the sigmoid functions are the most common choices ([161]) for these activation functions.

As far as the continuous-valued targets with a bounded range are concerned, one can again use the sigmoid functions, provided that either the outputs or the targets to

be scaled to the range of the output activation function. But if the target values have no known bounded range, it is better to use an unbounded activation function, most often the identity function (which amounts to no activation function).

In order to train a network and measure how well it performs, one also requires an objective function (or cost function) which must be defined to provide an unambiguous numerical rating of system performance. Selection of an objective function is very important because the function represents the design goals and decides what training algorithm can be taken. To develop an objective function that measures exactly what we want is not an easy task.

The next important concept is the network structures. The structure of a network is usually defined by the number of layers, the number of units per layer, and the interconnection patterns between layers. They are generally divided into two categories based on the pattern of connections: 'Feed-forward networks' and 'Recurrent networks'. In 'Feed-forward networks, the data flow from input units to output units is strictly feed-forward. The data processing can extend over multiple layers of units, but no feedback connections are present. However, in the 'Recurrent networks', which contain feedback connections, the dynamical properties of the network are important. Feed-forward neural networks are used in applications where the data does not contain time variant information while recurrent neural networks model time series sequences and possesses dynamical characteristics.

A layered feed-forward network consists of a certain number of layers, and each layer contains a certain number of units. There is an input layer, an output layer, and one or more hidden layers between the input and the output layer. Each unit receives its inputs directly from the previous layer (except for input units) and sends its output directly to units in the next layer (except for output units). Unlike the Recurrent network, which contains feedback information, there are no connections from any of the units to the inputs of the previous layers nor to other units in the same layer, nor to units more than one layer ahead. Every unit only acts as an input to the immediate next layer. Obviously, this class of networks is easier to analyze theoretically than other

general topologies because their outputs can be represented with explicit functions of the inputs and the weights.

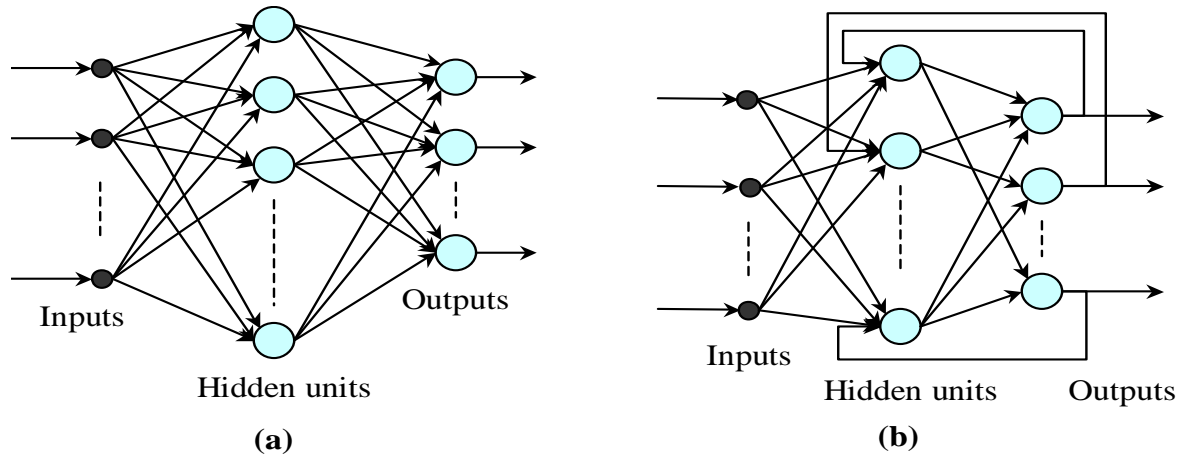


Figure 2.2.1: Architectures of feed-forward and recurrent neural networks

An example of a layered network with one hidden layer is shown in Figure 2.2.1.

The algorithm for evaluating the derivative of the error function is known as back-propagation, because it propagates the errors backward through the network. This back-propagation technique was popularized by [160]. It is the most commonly used method for training multi-layer feed-forward networks. It can be applied to any feed-forward network with differentiable activation functions.

In the learning process in most of the networks, one requires to use an error function which must be suitably minimized. This minimization has to be done with respect to the weights and bias. If a network has differentiable activation functions, then the input variable of the differentiable functions are the activations of the output units. These input variable are the weights and bias. One can evaluate the derivative of the error with respect to weights. These derivatives can then be used to find the weights that minimize the error function, by using the popular gradient descent method. For further details on this, as well as on the gradient descent algorithm, one may refer to [159].

It should be noted that the gradient descent optimization method is one of the most popularly used back-propagation learning algorithm. It has been proved as a

very successful method in many applications. However, this method does not converge very fast. Moreover, the convergence to the global minimum is not always guaranteed. Many researchers (see, e.g., [24, 70]) have attempted for improvements to the standard gradient descent method, such as dynamically modifying learning parameters or adjusting the steepness of the sigmoid function. To this end, gradient methods using second-derivatives (Hessian matrix), such as Newton's method, are found to be very efficient under certain conditions [153].

A large number of more sophisticated gradient descent algorithms have been developed (e.g., rprop, quickprop, conjugate gradients) that are generally observed to outperform steepest descent, if the gradient of the entire training set is calculated at once. Such methods are known as batch learning algorithms. However, one advantage of steepest descent is that it lends itself naturally to online learning (also known as sequential learning), where the weight updates are made after each pattern or sequences in the training set. Online steepest descent is often referred to sequential gradient descent or stochastic gradient descent.

Online learning tends to be more efficient than batch learning when large datasets containing significant redundancy or regularity are used [83]. In addition, the stochasticity of online learning can help to escape from local minima, since the stationary points of the objective function will be different for each training example. The stochasticity can be further increased by randomizing the order of the sequences in the training set before each pass through the training set (often referred to as a training epoch).

A recently proposed alternative for online learning is stochastic metadescent [81], which has been shown to give faster convergence and improved results for a variety of neural network tasks.

The generalization ability of neural networks is an important measure of their performance as it indicates the accuracy of the trained network when presented with data not present in the training set. A poor choice of network architecture will result in poor generalization even with optimal weight selection. The organization of neurons in the hidden layer may affect the generalization; too many neurons may result in overfitting

or overtraining while few neurons will result in underfitting.

The generalization performance in the case of overfitting may be improved by increasing the number of instances in the training set. Another technique is using weight decay during training. Weight decay in gradient descent learning fractionally decreases the weights at each iteration. A successful method is to provide a validation set in addition to the training data. In this method the training algorithm monitors the generalization error with respect to the validation set and terminates the training before the error increases.

It should be clear from the architecture of Feed-forward neural networks that past inputs have no way of influencing the processing of future inputs. This situation can be rectified by the introduction of feedback connections in the network. Now network activation produced by past inputs can cycle back and affect the processing of future inputs.

Feed-forward neural networks have been successfully used to solve problems that require the computation of a static function, i.e., a function whose output depends only upon the current input, and not on any previous inputs. In the real world however, one encounters many problems which cannot be solved by learning a static function because the function being computed changes with each input received.

As an example consider a deterministic finite state automata (DFA) which output its state every time it makes a transition. Clearly, the DFA may produce different outputs for the same input if it is in different states. Thus, any system that seeks to predict the outputs of a DFA must have some notion of how the past inputs affect the processing of the present input, as well as a way of storing the past inputs. In other words such a system must have a memory of the past input and a way to use that memory to process the current input.

The class of Neural Networks which contain cycles or feedback connections are called Recurrent Neural Networks (RNNs). While the set of topologies of a feed-forward networks is fairly constrained, an RNN can take on any arbitrary topology as any node in the network may be linked with any other node (including itself). The

only requirement we make is that the network have clearly defined input and output nodes. This we will discuss in next section.

## 2.3 Recurrent Neural Networks (RNNs)

In this section, we discuss about the advanced neural network architectures and its concepts have been studied extensively.

Recurrent neural networks are well suited for modeling time dependent processes due to their dynamical abilities. This feature has made them successful in applications to speech recognition, time series prediction, language learning and control. Recurrent neural networks are loose models of the brain and various architectures of recurrent neural networks have been applied to a wide range of problems with strengths and weaknesses in knowledge representation and learning capability. Gradient descent is most widely used for recurrent network training with variation in the form of backpropagation-through-time and real-time-recurrent learning. Symbolic knowledge in the form of finite automaton can be encoded in recurrent neural networks. Knowledge extraction from recurrent neural networks aims at finding the underlying models of the learnt knowledge in the form of finite state machines.

The feedback signals in recurrent neural networks may be transmitted through a time delay. In contrast to feed-forward networks, recurrent neural networks are dynamical systems whose next state and output depend on the present network state and input; they are particularly useful for modeling dynamical systems.

Although processing real-valued time series is much more popular, RNNs are also frequently used for symbolic sequence processing. For example in cognitive science community people often try to use recurrent neural networks to establish links between human ability to process linguistic structures and the potential of artificial RNNs [24, 154]. Other works study what kind of dynamical behaviour has to be acquired by RNNs to solve particular tasks such as processing strings of context-free languages, where counting mechanism is needed [70, 160].



Standard RNNs trained by common gradient descent techniques have problems with processing symbolic sequences containing long-time dependencies [55]. When propagating teaching signal, error tends to vanish or blow up. To overcome this limitation and to solve tasks, traditional RNNs cannot solve, novel architecture was suggested [55, 150]. Long short-term memory networks are equipped with special units called constant error carrousel with self-recurrent feedback connection of constant weight set to 1. In this way, error signal can span theoretically infinite time distances.

Another novel RNN architecture called Echo-state network (ESN) [103] is based on rich reservoir of potentially interesting behaviour. Reservoir is the RNNs recurrent layer formed by large number of sparsely interconnected units with non-trainable weights. Under certain conditions RNN state is a function of the finite history of inputs presented to the network - the state is the “echo” of the input history. Some of the architectures of first-order RNNs can be found in [157, 198] whereas those of the NARX networks can be found in [95, 117, 169]. For other type of networks, e.g., LSTM recurrent networks, Hopfield Networks, Fully Recurrent Neural Networks, Extended Kalman Filter and Decoupled Extended Kalman Filter Networks, one may refer to [74, 77, 91, 92, 103].

In Figure 2.3.1, we showed Recurrent networks architectures of Elman (figure a), Jordan (figure b), Robinson and Fallside (figure c) and (d) Williams and Zipser (figure d). Dashed lines indicate feedback connections. The Elman architecture used the context layer which makes a copy of the hidden layer outputs in the previous time steps. This architecture can be expanded to include additional hidden layers. The Jordan network is the earliest network specifically designed for temporal sequence modelling and gained popularity by highlighting the idea of context units. In the this architecture, the context layer receives feedback from the output as well as the context layer activations in the previous time steps. The context units are also referred to as state units. Hidden and output layer activations and respectively, are calculated at time  $t$ , from the weighted sum of the inputs and context units as follows:

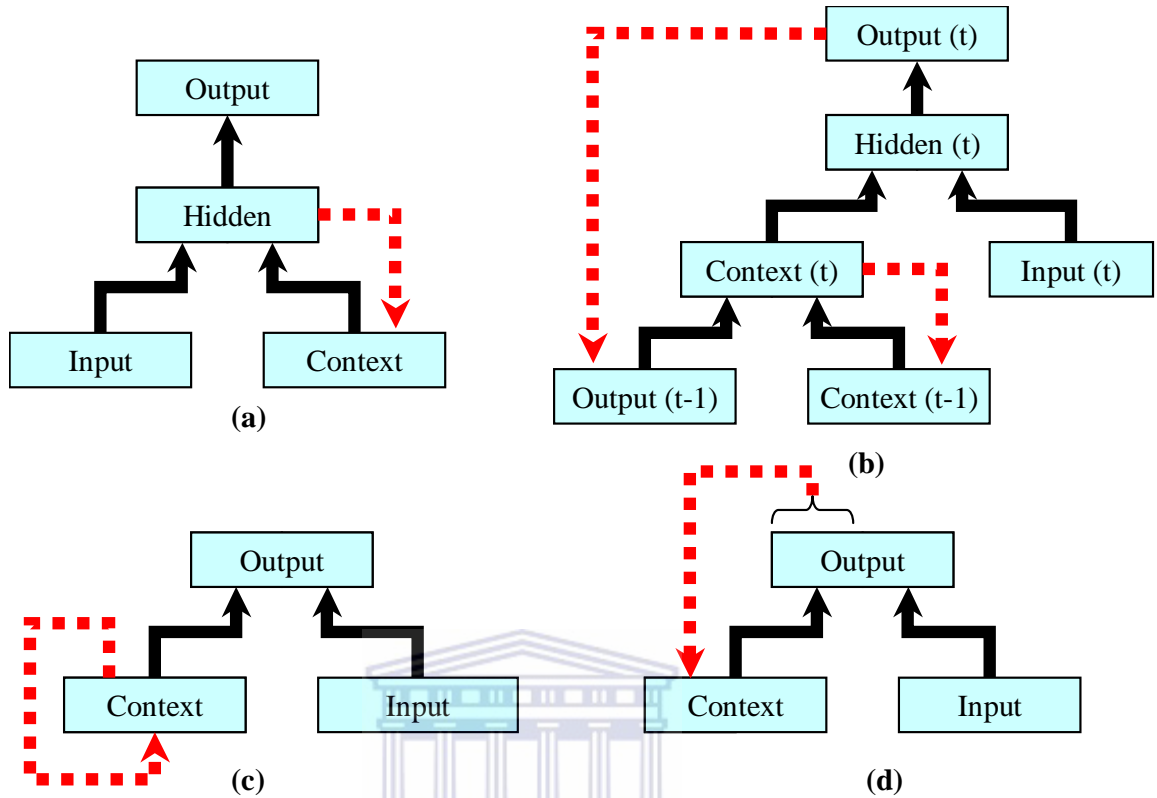


Figure 2.3.1: Architectures of RNN

$$h_j(t) = f \left( \sum_{i=1}^n w_{ji} x_i(t) + \sum_{i=1}^m w_{ji} c_i(t) \right).$$

$$y_k(t) = g \left( \sum_{j=1}^p w_{kj} h_j \right),$$

where  $n$ ,  $m$  and  $p$  are the number of input, context and hidden units, respectively. The context layer units are updated at each time step by

$$c_i(t+1) = \alpha c_i(t) + y_i(t).$$

The Robinson and Fallside architecture [155] is single layer network which has a context layer of fully recurrent connections. The model proposed by Williams and Zipser consists of a single layer of fully connected units from the output to the context layer in-

stead [198]. The outputs and context unit values at the following time step is computed by

$$y_j(t) = f \left( \sum_{i=1}^n w_{ji} x_i(t) + \sum_{i=1}^m w_{ji} c_i(t) \right),$$

$$c_i(t+1) = g \left( \alpha \sum_j w_{ij} y_j(t) \right).$$

### First-order recurrent neural networks:

The Jordan network has been used to model attractor dynamics [105] and is more suited to applications where the serial order is dependent on the previous context as well as the output. Applications of this network architecture include speech recognition and sunspot prediction [42, 105]. The most well-known application of the Elman network is for modeling the syntactic and semantic structure of English phrases [59, 60]. Other applications include financial data prediction [114] and Mackey-Glass time series prediction [42].

The model proposed by Mozer [130] has full adaptive connectivity between the input and context layer; the self-connections are also trained with other connections. Mozer derived a modified gradient learning rule by differentiating a cost function similar to that used by Williams and Zipser [198]. While this architecture has also been used for modeling finite-state behavior, it is more suitable for sequence classification than sequence generation. Both models by Jordan and Mozer employ what is termed *locally recurrent* or self connections. Applications of these architectures include prediction of future values of a financial time series [132] and process control [146].

The first-order recurrent neural network uses a context layer to store the output of the state neurons in the previous time steps. The context layer is used for computation of present states as they contain information about the previous states. They have shown to learn and represent deterministic finite automaton and fuzzy finite automaton [73].

In Figure 2.3.2, we present the architecture of a first order context layer RNN. The

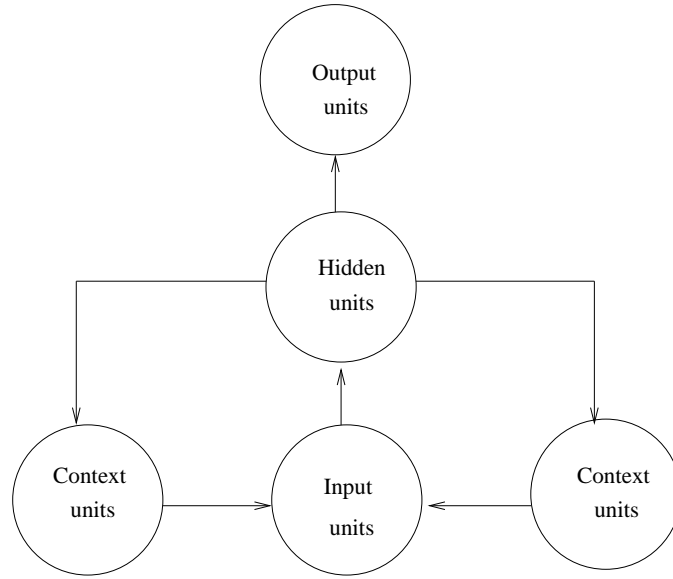


Figure 2.3.2: First order RNNs

connection from the context to hidden layer provides the recurrence relation. Note that the number of neurons in the hidden layer is equal to the number in the context layer. The neurons propagate information from one layer to another by computing a non-linear function of their weighted sum of inputs. The equation of the dynamics of the change of hidden state neuron activations in first order context layer networks is given by equation (2.3.1).

$$S_i(t) = g \left( \sum_{k=1}^K V_{ik} S_k(t-1) + \sum_{j=1}^J W_{ij} I_j(t-1) \right), \quad (2.3.1)$$

where  $S_k$  and  $I_j$  represent the output of the state neuron and input neurons respectively.  $V_{ik}$  and  $W_{ij}$  represent their corresponding weights, and  $g(\cdot)$  is a sigmoidal discriminant function.

Knowledge extractions through clustering and machine learning from first-order recurrent neural networks have been successfully applied [46]. They have been applied to a wide range of real world problems including speech recognition [155], financial price prediction [166] and gesture recognition [120].

**Second order recurrent neural networks:**

Goudreau [80] have shown that second-order single-layer recurrent neural networks are more suited to modeling finite-state behavior than their first-order counterparts. An experimental comparison of some recurrent network architectures on problems including grammatical inference and nonlinear systems identification can be found in [95].

The single-layer, second-order network employs product units of external input neurons  $x_k(t)$  and recurrent state units  $s_j(t)$  for all combinations of  $s_j(t) \times x_k(t)$  as input to the context units in the output layer. The activation for the context units is computed by:

$$s_i(t + 1) = f \left( \sum_{j=1}^n \sum_{k=1}^m w_{ijk} s_j(t) x_k(t) \right),$$

for  $n$  context units and  $m$  input units. This architecture has been widely applied to grammatical induction problems.

Second order recurrent neural networks are more suited for modelling finite-state behavior than first-order context layer networks [65]. However, it has been shown that first-order recurrent networks generalize better than second-order networks given that both of them have the same number of neurons as second-order networks have more weight connections [80]. The second-order recurrent network has been shown that deterministic finite automaton can be directly encoded in them. In Figure 2.3.3, we

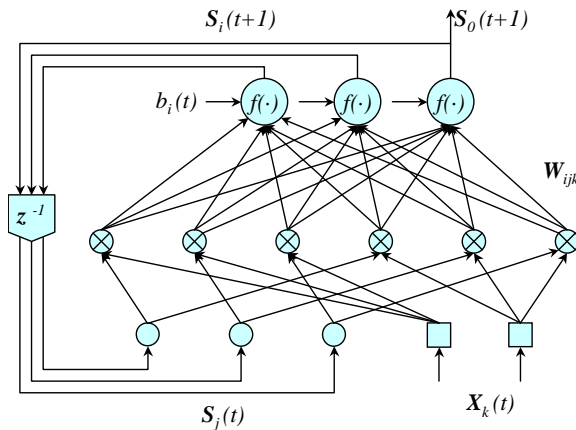


Figure 2.3.3: Second order RNNs

present the architecture of second-order RNNs. The second-order recurrent network with three state units and two input units. The output computes a sigmoidal function of weighted product units.

### Types of RNN architectures:

Here, we are giving a brief overview of different types of RNN architectures.

### Locally recurrent neural networks:

Locally recurrent neural networks are a class of networks that contain recurrent connections only within individual neurons [184]. It can be trained with gradient descent learning algorithms. It consists of an input layer, a layer of dynamic neurons with self-recurrent connections whose outputs are the inputs of a standard layered feed-forward network. The dynamics of locally recurrent neural networks is given as follows:

$$c_i(t) = f \left( w_{ii}^c c_i(t-1) + \sum_j w_{ij}^x x_j(t) \right),$$

$$y_i(t) = g \left( \sum_j w_{ij}^y c_j(t) \right),$$

where  $x_j(t)$ ,  $c_i(t)$  and  $y_i(t)$  represent the input, internal and output activations, respectively at time  $t$ . The self connection of the weight matrix is given by  $w_{ij}$  and  $f$  and  $g$  are the nonlinear functions.

### NARX recurrent neural networks:

NARX recurrent neural networks are inspired by nonlinear autoregressive models with exogenous inputs [169]. They have shown better training and generalization results when compared to other recurrent network architectures [95]. They compute their current output from past inputs and past outputs as shown in Figure 2.3.4. The architecture has two delay lines; one for input and the other for outputs with all taps

fully connected to the hidden layer. Their lengths are referred as input and output order, respectively.

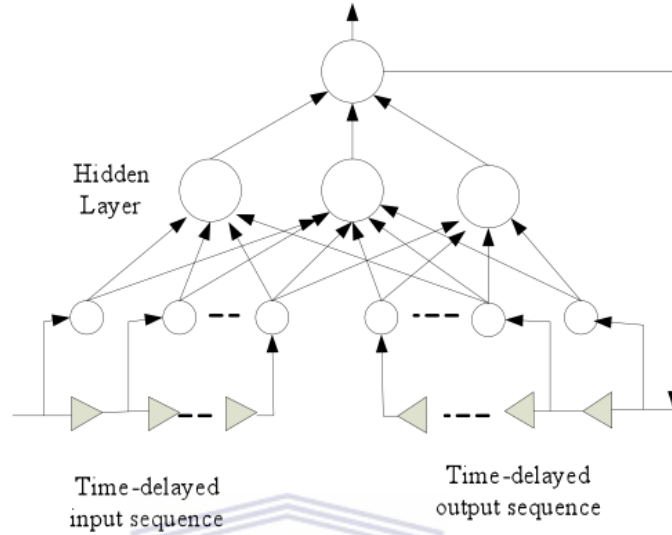


Figure 2.3.4: NARX RNNs

In Figure 2.3.4, we present the NARX network architecture. The NARX topology with two delayed lines for the inputs and outputs fully connected to the hidden layer. The hidden state neurons receive as inputs a window of past network input and output. The output  $y(t)$  for the linear case with input order  $T_x$  and  $T_x$  output order is given by

$$y(t) = \sum_{\tau=1}^{T_x} \alpha_{\tau} x(t - \tau) + \sum_{\tau=1}^{T_y} \beta_{\tau} y(t - \tau),$$

$$y(t) = f(x(t - T_x), \dots, x(t - 1), x(t), y(t - T_y), \dots, y(t - 1)).$$

NARX networks have been applied to finite automaton identification and signal processing [117]. They can retain information up to two or three times longer than conventional recurrent neural network architectures and hence can alleviate the problem of long-term dependencies [17].

### Long short term memory:

Recurrent neural networks can represent non-linear dynamical systems; however, learning long time dependencies can be difficult [17]. There is a difficulty in propagating long-time dependencies as the gradient descent learning mechanism can only reliably propagate information over certain time steps. In the process of error backpropagation, the error gradient approaches zero after certain number of time steps when  $n$  becomes large.

The Long Short Term Memory (LSTM) networks have been proposed to overcome the problem of long-term dependencies [92]. They have been applied to grammatical induction and speech recognition problems [75]. More recently, they have also been applied to bioinformatics problems with great success. They are composed of memory cells and gate units. Each memory cell is built around a central linear unit with a fixed self connection. The gate units open and close access to constant error carousel.

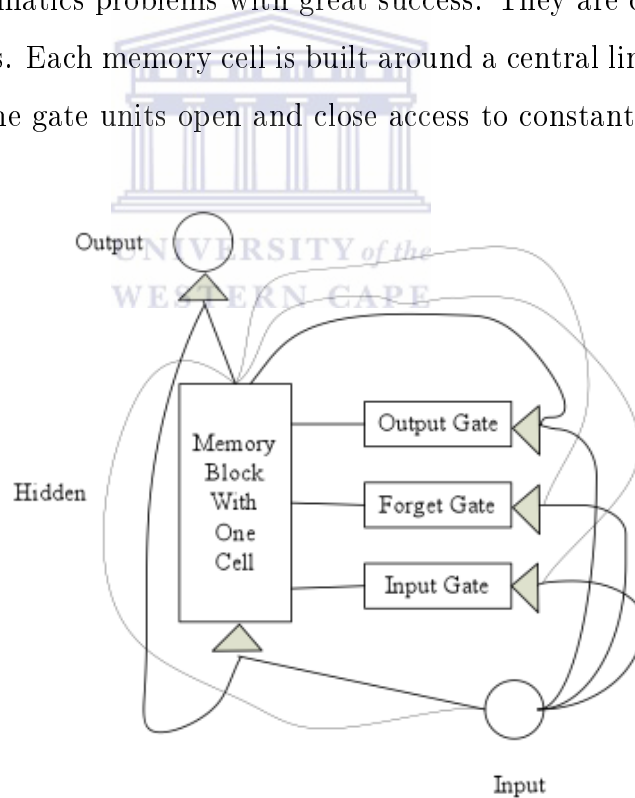


Figure 2.3.5: Long short term memory - RNN

LSTM networks can be trained using multi grid random search, time-weighted



pseudo Newton, discrete error backpropagation, and expectation maximization [18, 19]. LSTM solves complex long time lag tasks that have never been solved by previous recurrent network algorithms. It also works with local, distributed, real-valued, and noisy pattern representations.

The three learning algorithms for the training of Recurrent Neural Networks are as follows:

- Back Propagation Through Time [BPTT] algorithm,
- Real Time Recurrent Learning [RTRL] algorithm, and
- Decoupled Extended Kalman Filter [DEKF] algorithm.

### **Backpropagation through time:**

Backpropagation is the most widely applied learning algorithm for both feed-forward and recurrent neural networks. It learns the weights for a multilayer network, given a network with a fixed set of weights and interconnections. Backpropagation employs gradient descent to minimize the squared error between the networks output values and desired values for those outputs. The learning problem faced by backpropagation is to search a large hypothesis space defined by weight values for all the units of the network. Error is propagated from the output layer back to the hidden layers from which the weights are updated.

Backpropagation is used for training feed-forward networks while backpropagation through time (BPTT) is employed for training recurrent neural networks [197]. The BPTT is the extension of backpropagation algorithm. The general idea behind BPTT is to unfold the recurrent neural network in time so that it becomes a deep multilayer feed-forward network. This can be done by adding a layer for each time step. When unfolded in time, the network has the same behavior as a recurrent neural network for a finite number of time steps.

### Real Time Recurrent Learning:

Backpropagation-through-time uses the backward propagation of error information to compute the error gradient used in the weight update. An alternative approach for computing the gradient is to propagate the error gradient information forward. Real-time recurrent learning (RTRL) is a real time learning algorithm which updates the weights at the end of each sample string presentation with a gradient descent weight update rule. The algorithm computes the derivatives of states and outputs with respect to all weights as the network processes the sequence during the forward step [200]. There is no unfolding performed or necessary for real time recurrent learning.

In RTRL, the weights can be incremented on-line or at the end of the whole input sequence. Because on-line updating is possible, the RTRL algorithm can deal with input sequences of arbitrary length and does not require memory proportional to the length of input sequence. It allows recurrent networks to learn tasks that require retention of information over time periods having either fixed or indefinite length.

The recursive equation is given by

$$p_{ij}^k(t+1) = f_k(\text{net}_k(t)) \sum_{I \in U} w_{kI} p_{ij}^I(t) + \delta_{ik} z_j(t).$$

The weight update equations are

$$\Delta w_{ij}(t) = \mu \sum_{k \in U} e_k(t) p_{ij}^k(t)$$

and the overall correction to be applied to  $w_{ij}$  is given by

$$\Delta w_{ij}(t) = \sum_{t=t_0+1}^{t1} \Delta w_{ij}(t).$$

This avoids the need for allocating memory proportional to the maximum sequence length and leads to simple on-line implementations. The power of this method was demonstrated through a series of simulations [198].

### **Decoupled extended Kalman filtering:**

A standard Kalman filtering can be applied to linear system with Gaussian noise. Non-linear system such as RNNs with sigmoidal units, can be handled by extended Kalman filters (EKF). Decoupled variant of EKF [147, 199] is based on ignoring dependencies of weights feeding different units and thus significantly reduce computational requirements linked with the matrix inversion.

The following section describes about the hidden Markov models and its algorithms towards this study.

## **2.4 Hidden Markov Models (HMMs)**

In this section, the theoretical foundations and their algorithms of hidden Markov models has been discussed briefly. Particularly, we emphasized on the forward algorithm of HMM as it is the key link between neural networks and hidden Markov models.

Hidden Markov models were first described in a series of statistical papers by Baum and other authors in the second half of the 1960's. One of the first applications of HMM was speech recognition, starting in the mid 1970s. In the second half of the 1980's HMMs began to be applied to the analysis of biological sequences, in particular DNA. Since then, they have become ubiquitous in the field of Bioinformatics.

- Bioinformatics and Genomics
  - Prediction of protein-coding regions in genome sequences,
  - Modeling families of related DNA or protein sequences, and
  - Prediction of secondary structure elements from protein primary sequences.

A HMM is a finite set of states, each of which is associated with a probability distribution. Transitions among the states are governed by a set of probabilities called transition probabilities. In a particular state an outcome or observation can be generated, according to the associated symbol observation probability distribution. It is

only the outcome, not the state that is visible to an external observer and therefore states are “hidden” to the outside; hence the name HMM.

In a regular Markov model, the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a HMM, the state is not directly visible, but output, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states.

In order to define an HMM completely, the following elements are needed.

1. The number of states of the model,  $N$ .
2. The number of observation symbols in the alphabet,  $M$ . If the observations are continuous then  $M$  is infinite.
3. A set of state transition probabilities  $A = a_{ij}$ .
4. A probability distribution for the alphabets in each of the states  $B = b_j(k)$ .
5. If the observations are continuous then we will have to use a continuous probability density function, instead of a set of discrete probabilities.
6. The initial state distribution  $\pi_i$ .

Therefore we can use the compact notation

$$\lambda = (A, B, \pi),$$

to denote an HMM with discrete probability distributions.

### Three basic problems in HMMs:

Once we have an HMM, there are three problems of interest that we will consider below:

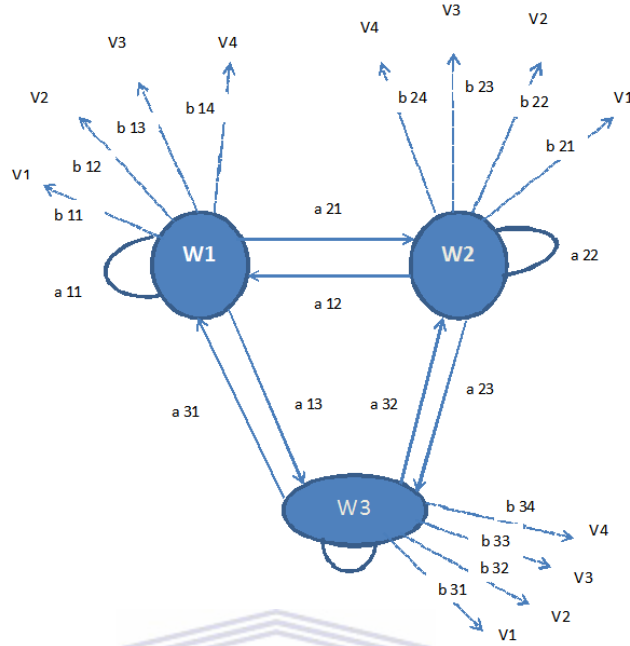


Figure 2.4.1: Hidden Markov model sample with emission states

1. The Evaluation Problem: Given an HMM  $\lambda$  and a sequence of observations  $O = \{o_1, o_2, \dots, o_T\}$  what is the probability that the observations are generated by the model,  $p(O|\lambda)$ ?

In the evaluation problem, namely given a model and a sequence of observations, how do we compute the probability that the observed sequence was produced by the model. We can also view the problem as one of scoring how well a given model matches a given observation sequence. The latter viewpoint is extremely useful. For example, if we consider the case in which we are trying to choose among several competing models, the solution to problem 1 allows us to choose the model which best matches the observations.

2. The Decoding Problem: Given the HMM  $\lambda = (\pi, A, B)$  and the observation sequence  $O = \{o_1, o_2, \dots, o_T\}$  calculate the most likely sequence of hidden states that produced this observation sequence  $O$ . Usually this problem is handled by Viterbi Algorithm.

3. The Learning Problem: Given a model  $\lambda$  and a sequence of observations  $O = \{o_1, o_2, \dots, o_T\}$  how should we adjust the model parameters  $(\pi, A, B)$  in order to maximize  $p\{O|\lambda\}$ . Usually this problem is handled by “Baum-Welch or Forward-backward algorithm/procedure”. As the forward algorithm of HMM exactly resembles the first order equation of Recurrent Neural Network, we are going to discuss the forward algorithm in detail for the evaluation problem.

As the forward algorithm of HMM exactly resembles the first order equation of RNN, we are going to discuss the forward algorithm in detail for the evaluation problem.

### The evaluation problem and the forward algorithm:

We have a model  $\lambda = (A, B, \pi)$  and a sequence of observations  $O = \{o_1, o_2, \dots, o_T\}$  and  $p\{O|\lambda\}$  must be found. We could calculate this quantity using simple probabilistic arguments. But the number of operations involved in this calculation is in the order of  $N^T$ . This is very large even if the length of the sequence,  $T$  is moderate. Therefore we adopt another method for this calculation which has considerably lower complexity and makes use of an auxiliary variable,  $\alpha_t(i)$  called forward variable.

The forward variable is defined as the probability of the partial observation sequence  $O = \{o_1, o_2, \dots, o_T\}$  when it terminates at the state  $i$ . Mathematically,

$$\alpha_t(i) = p\{o_1, o_2, \dots, o_t, q_t = i|\lambda\},$$

Then it is easy to see that following recursive relationship holds

$$\alpha_{t+1}(j) = b_j(o_{t+1}) \left[ \sum_{i=1}^N \alpha_t(i) a_{ij} \right], \quad 1 \leq j \leq N, \quad 1 \leq t \leq T - 1,$$

where

$$\alpha_1(j) = \pi_j b_j(o_1), \quad 1 \leq j \leq N.$$

Using this recursion we can calculate  $\alpha_T(i)$ ,  $1 \leq j \leq N$  which will be used in

$$p\{O/\lambda\} = \sum_{i=1}^N \alpha_T(i),$$

$$\alpha_t(i) = Pr(O_{1,\dots,t}, q_t = S_i).$$

The sum of the final  $\alpha$ 's is the probability of the observations.

$$\sum_i^N \alpha_T(i) = \sum_i^N Pr(O_{1,\dots,T}, q_T = S_i) = Pr(O_{1,\dots,T}).$$

The  $\alpha$ 's can be calculated as follows

$$\begin{aligned} \alpha_1(i) &= b_i(O_1)\pi_i, \\ \alpha_t(i) &= b_i(O_t) \sum_{j=1}^N a_{ji} \alpha_{t-1}(j). \end{aligned}$$

Same as  $\delta$  recursion except that max is replaced by sum.

The above forward algorithm can collectively be written as follows:

$$\begin{aligned}
\alpha_1(i) &= Pr(O_1, q_1 = S_i) \\
&= Pr(O_1 | q_1 = S_i) Pr(q_1 = S_i) \\
&= b_i(O_1) \pi_i \\
\alpha_t(i) &= Pr(O_{1,\dots,t}, q_t = S_i), \\
&= Pr(O_t | O_{1,\dots,t-1}, q_t = S_i) Pr(O_{1,\dots,t-1}, q_t = S_i), \\
&= Pr(O_t | q_t = S_i) Pr(O_{1,\dots,t-1}, q_t = S_i), \\
&= b_i(O_t) Pr(O_{1,\dots,t-1}, q_t = S_i), \\
&= b_i(O_t) \sum_{j=1}^N Pr(O_{1,\dots,t-1}, q_t = S_i, q_{t-1} = S_j), \\
&= b_i(O_t) \sum_{j=1}^N Pr(q_t = S_i | O_{1,\dots,t-1}, q_{t-1} = S_j) Pr(O_{1,\dots,t-1}, q_{t-1} = S_j), \\
&= b_i(O_t) \sum_{j=1}^N Pr(q_t = S_i | q_{t-1} = S_j) Pr(O_{1,\dots,t-1}, q_{t-1} = S_j), \\
&= b_i(O_t) \sum_{j=1}^N a_{ji} \alpha_{t-1}(j).
\end{aligned}$$

In the next chapter, the above forward calculation of HMM is used in the process of integrating and building the proposed hybrid HMM-RNN algorithm is presented, i.e., we will interpret the knowledge in HMM through RNN. Briefly, the hybrid HMM-RNN is the combination of first order calculation of RNN plus the forward algorithm of HMM.

## 2.5 Summary

We have briefly discussed the fundamental concepts and architectures of artificial neural networks, recurrent neural networks and hidden Markov models in this chapter. Feed-forward networks are applied in problems where time variant information is not present



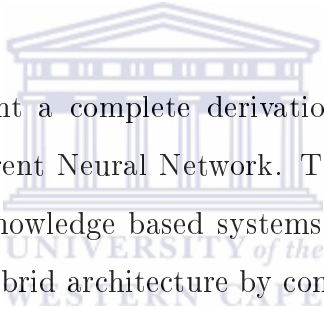
whereas recurrent neural networks are used for modeling dynamical processes. The goal of learning in neural networks is to approximate the output with a given set of inputs. Backpropagation employs gradient descent and is the most commonly used algorithm for training neural networks. The learning complexity and generalization are the two major factors which measure the performance of neural networks.

We have discussed the architectures of recurrent neural networks which include first-order recurrent networks, second-order recurrent networks, locally recurrent networks, NARX networks and LSTM networks. One limitation to neural networks is the difficulty to train using gradient descent learning where the network may get trapped in the local minima resulting in poor training and generalization performance. The extraction of finite-state automata from trained recurrent neural networks shows dynamical features of its knowledge representation.

In the following chapter, we will discuss the architecture and total framework of knowledge based systems using symbolic connectionist learning for the proposed hybrid HMM-RNN architecture. We will also present the mathematical derivation of this hybrid HMM-RNN architecture as well as the associated learning algorithm.

# Chapter 3

## Derivation and training of hybrid systems using HMM and RNN



In this chapter, we present a complete derivation of hybrid systems using Hidden Markov Model and Recurrent Neural Network. This chapter is the heart of this thesis. Here, we study the knowledge based systems framework using the connectionist systems. We design the hybrid architecture by combining the most powerful sequence recognition or classification algorithms. Finally, we present the mathematical derivation of this hybrid HMM-RNN architecture as well as the associated learning algorithm.

### 3.1 Introduction

In this section we will discuss about the introduction to the hybrid systems, relevant review on knowledge based systems, hybrid symbolic connectionist framework has been explained along with the the derivation of hybrid learning algorithm for HMM-RNN.

Hybrid systems combine strengths of at least two intelligent system paradigms. Examples of hybrid systems include, symbolic connectionist learning, evolutionary neural learning, neural expert systems and neuro-fuzzy systems. Neural expert systems combine the learning in neural networks with reasoning in expert systems. Therefore, they are able to learn from past experience and also have the ability to explain to their user

how they arrived to a particular solution. In hybrid symbolic connectionist learning, expert knowledge in neural networks is combined prior to training for better generalization and training performance. Neuro-fuzzy systems on the other hand combine human style reasoning of fuzzy logic with neural network learning. Evolutionary neural learning combines the optimization technique of genetic algorithm with neural networks to learn the weights of the network from past experience.

Below, we present an overview of the relevant literature on hybrid systems, knowledge based connectionist models, combinations of HMM and RNN in terms of hybrid systems in a chronological order.

In [29], Bridle explained the combination of recurrent neural network and hidden Markov model. In his model, RNN is treated as an HMM isolated word recognizer using full likelihood scoring for each word model. The author presented the units in the recurrent loop as linear, but the observations enter the loop via a multiplication. Training can use back-propagation of partial derivatives to hill-climb on a measure of discriminability between words. The back-propagation has exactly the same form as the backward pass of the Baum-Welch (EM) algorithm for maximum-likelihood HMM training. The author used the relative entropy error criterion (equivalent to the so-called Mutual Information criterion which has been used for discriminative training of HMMs) that have derivatives which are interestingly related to the Baum-Welch re-estimates and to Corrective Training.

The performance analysis of three evolutionary swarm computation technology-based methods, known as differential evolution (DE), particle swarm optimization (PSO), and the hybrid of DE and PSO (DEPSO), in training RNNs was investigated and presented by Rui *et al.* [158]. Furthermore, the gene networks are reconstructed via the identification of the gene interactions, which are explained through corresponding connection weight matrices. These authors studied and investigated because recurrent neural networks (RNNs) have attracted more efforts in inferring genetic regulatory networks (GRNs), using time series gene expression data from microarray experiments. This is critically important for revealing fundamental cellular processes, investigating

gene functions and understanding their relations. However, RNNs are well known for training difficulty. Traditional gradient descent-based methods are easily stuck in local minima and the computation of the derivatives is also not always possible. Their research results studied on two data sets which demonstrates the DEPSO algorithm performs better in RNN training. Also, the RNN-based model can provide meaningful insight in capturing the nonlinear dynamics of genetic networks and revealing genetic regulatory interactions.

A new hybrid model using discrete wavelet transform (DWT) and support vector machine (SVM) for distinguishing enzyme structures from non-enzymes was presented by Jianding *et al.* [104]. These authors studied because it is highly desirable to develop an automated method to identify whether a given new sequence belongs to enzyme or non-enzyme. These networks have been trained and tested on two datasets of proteins with different wavelet basis functions, decomposition scales and hydrophobicity data types. They obtained a maximum accuracy using SVM with a wavelet function of Bior 2.4, a decomposition scale  $j = 5$ , and Kyte-Doolittle hydrophobicity scales. Their validation and research results obtained by the self-consistency test, jackknife test and independent dataset test are encouraging, which indicates that the designed method can be employed as a useful assistant technique for distinguishing enzymes from non-enzymes.

In [138], Pandey and Mishra combined the Knowledge-based systems (KBS) and intelligent computing systems that are frequently used in the medical planning, diagnosis and treatment. Their KBS model consists of rule-based reasoning (RBR), case-based reasoning (CBR) and model-based reasoning (MBR) whereas intelligent computing method (ICM) encompasses genetic algorithm (GA), artificial neural network (ANN), fuzzy logic (FL) and others. The combination of methods in KBS such as CBR-RBR, CBR-MBR and RBR-CBR-MBR and the combination of methods in ICM is ANN-GA, fuzzy-ANN, fuzzy-GA and fuzzy-ANN-GA. The combination of methods from KBS to ICM is RBR-ANN, CBR-ANN, RBR-CBR-ANN, fuzzy-RBR, fuzzy-CBR and fuzzy-CBR-ANN. These authors studied and presented a different singular and combined

methods applicable to medical domain from mid 1970s to 2008. They represented in tabular form, showing that the methods and its salient features, processes and application areas in medical domain (diagnosis, treatment and planning). They observed that most of the methods are used in medical diagnosis very few are used for planning and moderate number in treatment.

In [44], Christos and Andreas presented a hybrid approach by combining the Self-Organizing Map (SOM) and the Hidden Markov Model (HMM). The Self-Organizing Hidden Markov Model Map (SOHMMM) establishes a cross-section between the theoretic foundations and algorithmic realizations of its constituents. These respective architectures and learning methodologies are fused in an attempt to meet the increasing requirements imposed by the properties of deoxyribonucleic acid (DNA), ribonucleic acid (RNA), and protein chain molecules. The fusion and synergy of the SOM unsupervised training and the HMM dynamic programming algorithms bring forth a novel on-line gradient descent unsupervised learning algorithm, which is fully integrated into the SOHMMM. Since the SOHMMM carries out probabilistic sequence analysis with little or no prior knowledge, it can have a variety of applications in clustering, dimensionality reduction and visualization of large-scale sequence spaces, and also, in sequence discrimination, search and classification. These authors conducted two series of experiments based on artificial sequence data and splice junction gene sequences and demonstrated the SOHMMM's characteristics and capabilities.

Sharma and Srinivasan [166] designed and explained a new hybrid model to examine the electricity price time series from dynamical system perspective and proposes a hybrid model which employs a synergistic combination of Recurrent Neural Network (RNN) and coupled excitable system for prediction of future prices in deregulated electricity markets. The approximation ability of Recurrent Neural Networks to map dynamic functions together with sharp jumping attribute of coupled excitable systems allows close approximation of spiky time series. These authors developed hybrid model, which was applied for point and interval forecasting in various markets worldwide over different seasons for testing its adaptability in different environments. Their final

results of the prediction were obtained in all the markets, in stable as well as spiking regions of the time series.

A new hybrid model for integrating neural network language models in the decoding process of three state-of-the-art systems: one based on bidirectional recurrent neural networks, another based on hybrid hidden Markov models and, finally, a combination of both was given by Zamora *et al.* [204]. These authors examined due to unconstrained off-line continuous handwritten text recognition is a very challenging task which has been recently addressed by different promising techniques. The validation results obtained on the IAM off-line database demonstrate that consistent word error rate reductions can be achieved with neural network language models when compared with statistical N-gram language models on the three tested systems. The best word error rate 16.1% reported with ROVER combination of systems using neural network language models significantly outperforms current benchmark results for the IAM database.



### Symbolic connectionist learning:

The general paradigm of symbolic connectionist learning includes the combination of symbolic knowledge in neural networks for better training and generalization performance [1, 67]. The traditional connectionist representation approach of using neural networks includes initializing of neural network with small random values and training it using some optimization methods such as gradient descent and genetic algorithms on some known data to perform a certain task. After successful training, the network can take advantage of its generalization capability to perform tasks such as classification and recognition when presented with data. During the entire process, the knowledge remains hidden in the networks adaptable connections, hence the name connectionist representation. The connectionist representation is shown in the Figure 3.1.1.

The paradigm in the connectionist representation can be enriched with symbolic knowledge by initializing a network with prior knowledge, i.e., the initial domain theory,

prior to training. A translation of information from a symbolic into a connectionist representation is required. This is done by programming subset of weights in the network prior to training instead of choosing small random values. The programmed weights define a starting point in weight space for a search of a solution during training. Examples of this approach include pre-structuring a network with boolean concepts and imposing rotation variance in neural networks for image recognition [13, 67, 181].

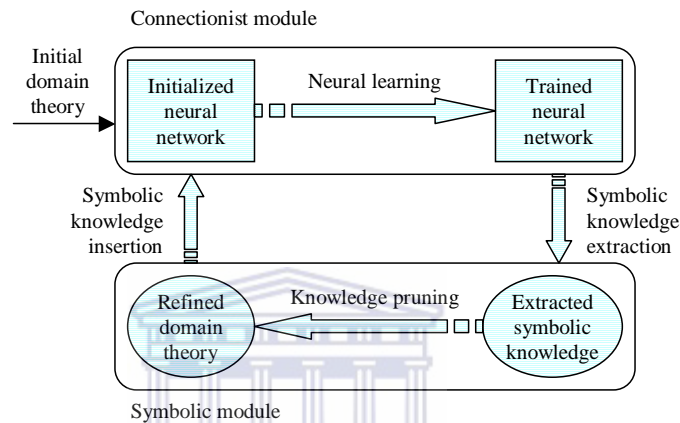


Figure 3.1.1: Knowledge based neural network

In Figure 3.1.1, we present a framework for combining symbolic and neural learning. The use of neural networks for knowledge refinement consists of (i) insertion of prior knowledge known as initial domain theory into a neural network, (ii) refinement of knowledge through training a network on examples, and (iii) extraction of learnt knowledge from a trained network in symbolic form, known as refined domain theory. Once the network has been trained, knowledge can be extracted in symbolic form, i.e., the refined domain theory. The extracted knowledge may approximate the networks true knowledge; in some cases the extracted knowledge may outperform the performance of the trained network.

### The significance and insertion of prior knowledge:

The fidelity in the mapping of the prior knowledge is very important since the network may not take advantage of poorly encoded knowledge. Poorly encoded knowledge may

hinder the learning process. Good prior knowledge encoding may provide the network with beneficial features such as

- The learning process may lead to faster convergence to a solution meaning better training performance,
- the networks trained with prior knowledge may provide better generalization when compared to networks trained with no prior knowledge and,
- the rules in prior knowledge may help to generate additional training data which are not present in the original data set.

Prior knowledge usually represented in the form of explicit rules in symbolic form is encoded in neural networks by programming some weights prior to training [181]. In feed-forward neural networks, prior knowledge is encoded in propositional logic expression form by programming a subset of weights. Prior knowledge also determines the topology of the network, i.e., the number of neurons and hidden layers appropriate for encoding the knowledge. The paradigm has been successfully applied to real world problems including bio-conservation and molecular biology. The prior or expert knowledge helps the network to get better generalization and training performance when compared with network architecture with no prior knowledge encoding.

For recurrent neural networks, finite-state automata are the basis for knowledge insertion. It has been shown that deterministic finite-state automata can be encoded in discrete-time second-order recurrent neural networks by directly programming a small subset of available weights. For first order recurrent neural networks, a method for encoding finite-state automata has been proposed and shown in [66].

### **Knowledge extraction:**

In the past, neural networks were considered as black boxes as they were not able to explain the knowledge acquired in the weights after the training process. Research on this topic has resulted in a number of algorithms for knowledge extraction in symbolic



form [46]. In feed-forward networks, knowledge is usually extracted in the form of Boolean and fuzzy if-then clauses [69, 86]. For recurrent networks, finite-state automata have been the main paradigm for temporal symbolic knowledge extraction [45].

The goal of knowledge extraction is to find the knowledge stored in the network weights in symbolic form. One main concern is the fidelity of the extraction process, i.e., how accurately the extracted knowledge corresponds to the knowledge stored in the network. Extraction algorithms can be basically divided into two classes: decompositional methods extract rules from the internal networks structure by looking at the weights and nodes of the network, e.g., extraction through clustering [46]. Pedagogical methods view trained neural networks as black-boxes and uses some machine learning methods to extract rules obtained from the input-output mapping of the network e.g machine learning TB algorithm.

### **Knowledge refinement:**

Knowledge refinement or revision is the main goal of learning in a hybrid system where neural learning together with knowledge extraction is combined to produce a more accurate set of rules within a given domain [167]. The initial domain knowledge, which may also contain information inconsistent with the available training data, is encoded in a network. The network is then trained with the available data set with several training runs depending on how close the initial symbolic knowledge is to the final solution. Then the refined or revised rules, i.e., in case of poor prior knowledge can be extracted from the trained network in symbolic form. The benefits of knowledge refinement include

- Better training performance,
- Improved generalization performance, and
- Clear understanding of the internal representation of the trained network.

The rest of the chapter is organized as follows. We describe recurrent neural networks based on the hidden Markov models in Section 3.2. Real time recurrent learning for the hybrid HMM-RNN system is presented in Section 3.3. Some applications of hybrid systems are explored in Section 3.4. Finally, a brief summary is provided in Section 3.5.

## 3.2 RNNs based on HMMs

In this section, we present a link between recurrent neural networks and hidden Markov models, significance of the hybrid system, derivation of the algorithm.

Recurrent neural networks (RNNs) have been an important focus of research as they can be applied to difficult problems involving time-varying patterns. Their applications range from speech recognition and financial prediction to gesture recognition [155]. They have the ability to provide good generalization performance on unseen data but are difficult to train. Hidden Markov models (HMMs), on the other hand, have also been applied to solve difficult real world problems involving time-varying patterns. For instance, they have been very popular in areas of speech recognition [71]. Training HMMs is easy, i.e., they learn faster when compared to recurrent neural network, but their generalization performance may not perform satisfactorily when compared to the performance of recurrent neural networks.

The structural similarities between HMMs and RNNs are the basis for mapping HMMs into RNNs. The combination of the two paradigms into a hybrid system may provide better generalization and training performance which would be a useful contribution to the field of machine learning and pattern recognition. We call the new hybrid architecture as hybrid HMM-RNN in further discussions.

### **Significance of hybrid HMM-RNN:**

We have stated earlier that the structural similarities of hidden Markov models and recurrent neural networks form the basis for combining the two paradigms into a hybrid

architecture. Why is it a good idea? Most often, first-order HMMs are used in practice in which successor states are dependent only on the previous state. This assumption is unrealistic for many real world applications of HMMs. It has been shown that RNNs can learn higher-order dependencies from training data [20]. Furthermore, the number of states in the HMMs needs to be fixed beforehand for a particular application. However, the numbers of states for different applications vary. The theory on RNNs and HMMs suggest that the combination of the two paradigms may provide better generalization and training performance. Our proposed architecture of hybrid recurrent neural networks may also have the capability of learning higher order dependencies and one does not need to fix the number of states as in the case of HMMs.

### Derivation and training of hybrid HMM-RNN:

Here, we presented the structural similarities of the two paradigms and design the hybrid recurrent neural networks architecture. Consider the equation of the forward procedure for the calculation of the probability of the observation  $O$  given the model, thus in HMM is given by

$$\alpha_j^t = \left( \sum_i^N \alpha_i^{t-1} a_{ij} \right) \cdot b_j(o^t), \quad (3.2.1)$$

where  $N$  is the number of hidden states in the HMM,  $a$  is the probability of making a transition from state  $i$  to  $j$  and  $b_j(o^t)$  is the Gaussian distribution for the observation at time  $t$ . The calculation in equation (3.2.1) is inherently recurrent and bares resemblance to the recursion of recurrent neural networks as shown in equation (3.2.2)

$$x_i^t = f \left( \sum_j^N x_j^{t-1} w_{ji} \right), \quad (3.2.2)$$

where  $f(\cdot)$  is a non-linearity as sigmoid,  $N$  the number of hidden neurons and  $w_{ji}$  the weights connecting the neurons with each other and with the input nodes. We are now

going to consider the dynamics of first-order recurrent neural network which is given by

$$y_j(t) = f \left( \sum_{i=1}^N w_{ji}x_i(t) + \sum_{i=1}^M w_{ji}c_i(t) \right). \quad (3.2.3)$$

$$c_i(t+1) = g \left( \sum_j w_{ij}y_j(t) \right), \quad (3.2.4)$$

where  $w_{ji}$  represent their corresponding weights and  $g(\cdot)$  is a sigmoidal discriminant function.

Let us combine equation (3.2.1) with equations (3.2.2) and (3.2.4) to form a hybrid architecture. We are replacing the subscript  $j$  in  $b_j(\sigma^t)$  which denotes the state by time  $t$  in hidden Markov models - to incorporate the feature into recurrent neural networks. Hence, the dynamics for the hybrid recurrent neural networks architecture is given by

$$y_j(t) = f \left( \sum_{i=1}^N w_{ji}x_i(t) + \left[ \sum_{i=1}^M w_{ji}c_i(t) \right] b_{t-1}(O) \right), \quad (3.2.5)$$

where  $b_{t-1}(O)$  is the Gaussian distribution. Note that the subscript in  $b_{t-1}(O)$ , i.e., time  $t$ , in equation (3.2.5) is different from the subscript for Gaussian distribution in equation (3.2.1). The dynamics of hidden Markov models and recurrent networks varies in this context; however, we can adjust the parameter for time  $t$  as shown in equation (3.2.5) in order to map hidden Markov models into recurrent neural networks. For a single input, the univariate Gaussian distribution is given by

$$b_t(O) = \frac{1}{\sqrt{2\pi\sigma}} \exp \left( -\frac{1}{2} \frac{(O - \mu)^2}{\sigma^2} \right), \quad (3.2.6)$$

where  $O$  is the observation at time  $t$ ,  $\mu$  is the mean and  $\sigma_i^2$  is the variance. Similarly, for the discrete inputs, the Gaussian Distribution using histograms is defined as

$$\text{Observation at time } t = \frac{\text{Number of observations occurred at time } t}{\text{Total number of observations}}. \quad (3.2.7)$$

Finally, the observation probabilities for discrete case is calculated as frequency of these inputs.

For Continuous inputs, we define the Gaussian Mixture Models which is a parametric probability density function and represented as weighted sum of Gaussian component densities and it is given by the equation:

It can also be represented in the following form

$$b_j(o_t) = \sum_{m=1}^M C_{jm} \mathcal{N}(o_t; \mu_{jm}, \Sigma_{jm}), \quad (3.2.8)$$

where  $(x, \mu, U)$ , denotes a  $D$ -dimensional normal density function of mean vector  $\mu$  and covariance matrix  $U$  and  $M$  is the number of mixture components for the distribution, and  $C_{jm}$ ,  $\mu_{jm}$  and  $\Sigma_{jm}$  are a weight, a  $L$ -dimensional mean vector, and a  $L\{x\}L$  covariance matrix of mixture component  $m$  of state  $i$ , respectively.

Mixture weights  $C_{jm}$  satisfy the following stochastic constraint  $\sum_{m=1}^M C_{jm} = 1$ , Hence,  $b_i(o)$ 's are normalized as probability density function.

A Gaussian distribution  $(o, \mu_{jm}, \Sigma_{jm})$  of each component is defined by

$$b_t(O) = \frac{1}{2\pi^{d/2} |\Sigma|^{1/2}} \exp \left[ -\frac{1}{2} (O - \mu)^t \Sigma^{-1} (O - \mu) \right], \quad (3.2.9)$$

where  $O$  is a  $d$ -component column vector,  $\mu$  is a  $d$ -component mean vector,  $\sigma$  is a  $d \times d$  covariance matrix, respectively.

Figure 3.2.1 shows how the Gaussian distribution for hidden Markov model is mapped into hybrid recurrent neural networks. The output of the Gaussian function solely depends on the two input parameters which are the mean and the variance. These are parameters that observe the sequence of the input data in the hybrid architecture which may be DFA strings or data from any real-world time series for example bioinformatics, biometrics and etc..

In Figure 3.2.1, we present the architecture of a hybrid HMM-RNN. The dashed lined indicates that the architecture can represent more neurons in hidden and input

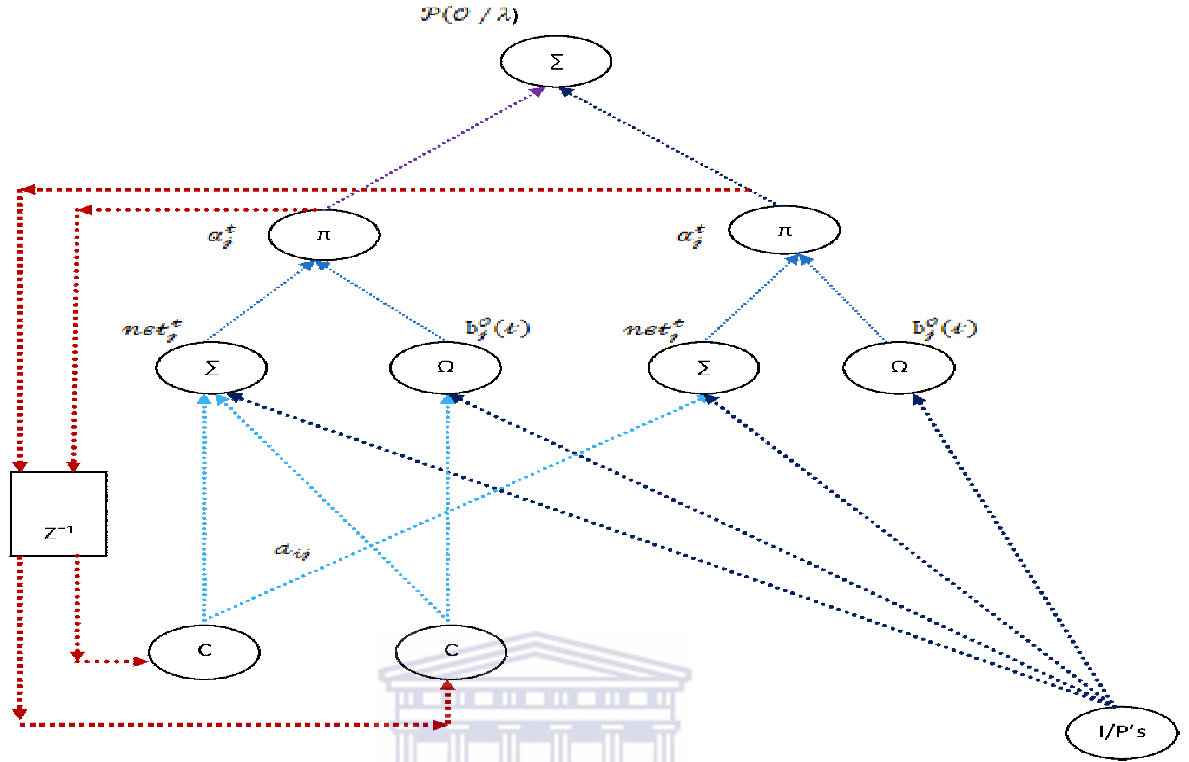


Figure 3.2.1: Hybrid HMM-RNN Architecture

layer if required. The output of the Gaussian is further multiplied with the output of the neurons in the hidden layer. Note that one Gaussian distribution will be used irrespective of the number of neurons in hidden and input layer.

Hence, by combining and representing the forward algorithm of HMM equation (3.2.1) in terms of RNN equation (3.2.2) and taking the derivative with respect to  $a_{ij}$ , we get

$$\begin{aligned}
 \frac{\partial}{\partial a_{ij}}(\alpha_j^t) &= \left[ \sum_{i=1}^N \frac{\partial}{\partial a_{ij}} (\alpha_i^{t-1} a_{ij}) b_j(o^t) \right] \\
 &= \left[ \sum_{i=1}^N \frac{\partial}{\partial a_{ij}} (\alpha_i^{t-1} a_{ij}) \right] b_j(o^t) \\
 &= \left[ \sum_{i=1}^N \frac{\partial}{\partial a_{ij}} (\alpha_1^{t-1} a_{1j} + \dots + \alpha_n^{t-1} a_{nj}) \right] b_j(o^t). \quad (3.2.10)
 \end{aligned}$$

Now

$$\begin{aligned} \frac{\partial}{\partial a_{1j}}(\alpha_1^{t-1} a_{1j}) &= \left[ \frac{\partial}{\partial a_{1j}} (\alpha_1^{t-1} a_{1j}) \frac{\partial a_{1j}}{\partial a_{ij}} \right] b_j(o^t), \\ \Rightarrow \frac{\partial}{\partial a_{1j}}(\alpha_1^{t-1} a_{1j}) &= \begin{cases} \frac{\partial}{\partial a_{1j}} (\alpha_1^{t-1} a_{1j}) 1 & \text{if } i = 1 \\ 0 & \text{if } i \neq 1, \end{cases} \\ \Rightarrow \frac{\partial}{\partial a_{1j}}(\alpha_1^{t-1} a_{1j}) &= \frac{\partial}{\partial a_{1j}}(\alpha_1^{t-1} a_{1j}) \\ &= \alpha_1^{t-1} \frac{\partial}{\partial a_{1j}} a_{1j} \\ &= \alpha_1^{t-1}. \end{aligned}$$

Similarly,

$$\frac{\partial}{\partial a_{2j}}(\alpha_2^{t-1} a_{2j}) = \alpha_2^{t-1}.$$

Hence

$$\frac{\partial}{\partial a_{ij}}(\alpha_j^t) = [\alpha_1^{t-1} + \alpha_2^{t-1} + \dots + \alpha_N^{t-1}] b_j(o^t),$$

$$\begin{aligned} \frac{\partial}{\partial a_{ij}}(\alpha_j^t) &= \left[ \sum_{i=1}^N \frac{\partial}{\partial a_{ij}} (\alpha_i^{t-1} a_{ij}) \right] b_j(o^t) \\ &= \left[ \sum_{i=1}^N \alpha_i^{t-1} \right] b_j(o^t). \end{aligned} \tag{3.2.11}$$

In the case of training strings of certain lengths representing finite automaton, a univariate Gaussian for one dimensional input will be used as shown in equation (3.2.6). For real world applications where multiple dimensions are involved, multivariate Gaussian function would be used instead as shown in equation (3.2.8).

In gradient descent training algorithm, the general real time recurrent learning of recurrent neural networks [200] can be applied keeping in mind that additional parameters, i.e., the mean and variance of the Gaussian distribution, must be trained.

Now before we move onto the next section, which deals with the real time recurrent

learning algorithm for the derived HMM-RNN system in this thesis, we would like to mention that some other relevant works are appropriately reviewed in the individual sections.

### 3.3 Real time recurrent learning for the hybrid HMM-RNN system

In this section, we presented the RTRL algorithm for the developed HMM-RNN architecture along with a brief overview of the computational models.

We presented a learning algorithm which is based on the Williams and Zipser's real time recurrent learning (RTRL) algorithm [198]:

In deriving a gradient-based update rule for recurrent networks, we make network connectivity highly unconstrained. We simply suppose that we have a set of input units,  $I = x_k(t), 0 < k < m$ , and a set of other units,  $U = y_k(t), 0 < k < n$ , which can be hidden or output units. To index an arbitrary unit in the network we can use

$$z_k(t) = \begin{cases} x_k(t) & \text{if } k \in I, \\ y_k(t) & \text{if } k \in U. \end{cases} \quad (3.3.1)$$

Let  $W$  be the weight matrix with  $n$  rows and  $n+m$  columns, where  $w_{ij}$  is the weight to unit  $i$  (which is in  $U$ ) from unit  $j$  (which is in  $I$  or  $U$ ). Units compute their activations in the now familiar way, by first computing the weighted sum of their inputs:

$$\text{net}_k(t) = \sum_{I \in U \cup I} w_{kI} z_I(t), \quad (3.3.2)$$

$$\text{net}_k(t) = \sum_{I \in U \cup I} w_{kI} z_I(t) b_t(O). \quad (3.3.3)$$

Here  $b_t(o)$  got two cases:



**Case - 1:**

$$b_t(O) = \frac{1}{\sqrt{2\pi\sigma}} \exp \left[ -\frac{1}{2} \frac{(O - \mu)^2}{\sigma^2} \right]. \quad (3.3.4)$$

**Case - 2:**

$$b_t(O) = \frac{1}{2\pi^{d/2} |\Sigma|^{1/2}} \exp \left[ -\frac{1}{2} (O - \mu)^t \Sigma^{-1} (O - \mu) \right], \quad (3.3.5)$$

where the only new element in the formula is the introduction of the temporal index  $t$ . Units then computes some non-linear function of their net input

$$y_k(t+1) = f_k [net_k(t)] \quad (3.3.6)$$

Usually, both hidden and output units will have non-linear activation functions. Note that external input at time  $t$  does not influence the output of any unit until time  $t+1$ . The network is thus a discrete dynamical system.

Let  $T(t)$  be the set of indices  $k$  in  $U$  for which there exists a target value  $d_k(t)$  at time  $t$ . We are forced to use the notation  $d_k$  instead of  $t$  here, as  $t$  now refers to time. Let the error at the output units be

$$e_k(t) = \begin{cases} d_k(t) - y_k(t) & \text{if } k \in T(t), \\ 0 & \text{otherwise.} \end{cases}$$

and define our error function for a single time step as

$$E(\tau) = -\frac{1}{2} \sum_{k \in U} [e_k(\tau)]^2.$$

The error function we wish to minimize is the sum of this error over all past steps of the network

$$E_{total}(t_0, t_1) = \sum_{\tau=t_0+1}^{t_1} E(\tau).$$

Now, because the total error is the sum of all previous errors and the error at this time step, so also, the gradient of the total error is the sum of the gradient for this time step and the

gradient for previous steps

$$\nabla_w E_{total}(t_o, t+1) = \nabla_w E_{total}(t_o, t) + \nabla_w E(t+1).$$

As a time series is presented to the network, we can accumulate the values of the gradient, or equivalently, of the weight changes. We thus keep track of the value

$$\Delta w_{ij}(t) = -\mu \frac{\partial E(t)}{\partial w_{ij}}.$$

After the network has been presented with the whole series, we alter each weight  $w_{ij}$  by

$$\sum_{t=t_0+1}^{t1} \Delta w_{ij}(t). \quad (3.3.7)$$

Then we compute

$$-\frac{\partial E(t)}{\partial w_{ij}} = -\sum_{k \in U} \frac{\partial E(t)}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial w_{ij}} = \sum_{k \in U} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}},$$

at each time step  $t$ . Since we know  $e_k(t)$  at all times (the difference between our targets and outputs), we only need to find a way to compute the second factor  $\frac{\partial y_k(t)}{\partial w_{ij}}$ .

Derivation of  $\frac{\partial y_k(t)}{\partial w_{ij}}$ :

From equation (3.3.6) and equation (3.3.7) we get

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = f_k[net_k(t)] \left[ \sum_{I \in U \cup I} w_{kI} \frac{\partial y_I(t)}{\partial w_{ij}} + \delta_{ik} z_j(t) \right], \quad (3.3.8)$$

where  $\delta_{ik}$  is the Kronecker delta

$$\delta_{ik} = \begin{cases} 1 & \text{if } i = k, \\ 0 & \text{otherwise.} \end{cases}$$

Because input signals do not depend on the weights in the network,

$$\frac{\partial y_I(t)}{\partial w_{ij}} = 0 \text{ for } i \in I,$$

equation (3.3.8) becomes

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = \left[ f_k(\text{net}_k(t)) \left[ \sum_{I \in U \cup I} w_{kI} \frac{\partial y_I(t)}{\partial w_{ij}} \right] b_t(O) + \delta_{ik} z_j(t) \right]. \quad (3.3.9)$$

This is a recursive equation. Because we have assumed that our starting state ( $t = 0$ ) is independent of the weights, then we have

$$\frac{\partial y_k(t_0)}{\partial w_{ij}} = 0.$$

These equations hold for all. We therefore need to define the values

$$p_{ij}^k(t) = \frac{\partial y_k(t)}{\partial w_{ij}},$$

for every time step  $t$  and all appropriate  $i, j$  and  $k$ . We start with the initial condition

$$p_{ij}^k(t) = 0, \quad (3.3.10)$$

and compute at each time step along by substituting the equation (3.2.11), we get

$$p_{ij}^k(t+1) = \left[ f_k(\text{net}_k(t)) \left[ \sum_{I \in U} w_{kI} p_{ij}^I(t) \right] b_t(O) + \delta_{ik} z_j(t) \right]. \quad (3.3.11)$$

The algorithm then consists of computing, at each time step  $t$ , the quantities  $p_{ijk}(t)$  using equations (3.3.10) and (3.3.11) and then using the differences between targets and actual outputs to compute weight changes

$$\Delta w_{ij}(t) = \mu \sum_{k \in U} e_k(t) p_{ij}^k(t), \quad (3.3.12)$$

and the overall correction to be applied to  $w_{ij}$  is given by

$$\Delta w_{ij}(t) = \sum_{t=t_0+1}^{t_1} \Delta w_{ij}(t). \quad (3.3.13)$$

Hybrid HMM-RNN can be trained both with gradient descent learning methods and

genetic algorithms. In the case of training strings of certain lengths representing finite automaton, a univariate Gaussian for one dimensional input will be used as shown in equation (3.2.6). For real world applications, where multiple dimensions are involved, multivariate Gaussian function is used instead of equation shown in (3.2.8).

In gradient descent training, the general backpropagation through time learning can be applied keeping in mind that additional parameters, i.e., the mean and variance of the Gaussian distribution, must be trained. Gradient descent learning, as discussed earlier, has its drawbacks as the network can become trapped in the local minima resulting in poor training and generalization performance.

### **Recurrent neural networks as models of computation:**

Recurrent neural networks are appropriate tools for modeling time varying systems for example; speech recognition, physical dynamical systems, and financial prediction. However, these applications are not well suited for addressing their fundamental issues such as training algorithms and knowledge representation. These applications come with specific characteristics, for example, in application to speech recognition feature extraction may be required which may hinder the investigation of networks fundamental issues. Different applications require different feature extraction techniques.

The models such as finite-state automata and their corresponding languages can be viewed as a general paradigm of temporal, symbolic language. There is no feature extraction necessary for recurrent neural networks to learn these languages. The knowledge acquired in recurrent neural networks through learning well corresponds with the dynamics of finite-state automata. The representation of automata as a prerequisite for learning its corresponding languages; i.e., if the architecture cannot represent a particular automaton then it would not be able to learn it either.

## **3.4 Applications of hybrid systems**

In this section, we presented some of the key applications of hybrid systems has been discussed and concluded the chapter with the overall summary.

### **Speech recognition:**

Speech recognition systems are composed of two major components. These are (1) feature extraction component which extracts features from a speech database, and (2) machine learning component which builds a model on the extracted features. A speech sequence contains huge amount of irrelevant information. In order to model them, feature extraction is necessary. In feature extraction, useful information from speech sequences are extracted which then is used for modeling using recurrent neural networks.

Hybrid systems have been successfully applied in modeling speech sequences for large vocabulary speech recognition problems [29]. They have been applied to isolated word recognition. The performance of speech recognition system can be measured in terms of accuracy and speed. They have been applied to recognize words and phonemes. Extensive research on the application of research recognition has been done for more than forty years; however, scientists are unable to implement systems which can show excellent performance in environments with background noise. Applications of speech recognition include voice command systems in home ware devices, speech input devices for interaction with computers other than mouse and keyboards, and speech command interaction with robots.

### **Molecular Biology:**

The prediction of three-dimensional structure of proteins is an important problem in molecular biology. The tertiary structure determines the function of a protein. Direct determination of the tertiary structure using methods such as X-ray crystallography is expensive and time consuming. The local or secondary structure of proteins provides good approximation of the three-dimensional structure also known as strings of amino acids. The Chou-Fasman algorithm is the standard algorithm for the problem which achieves a prediction accuracy of up to 58 percent. Recurrent neural networks have been initiated with Chou-Fasman domain and have been trained using sliding windows of amino acid sequences [51]. Knowledge based neuro-computing paradigm using feed-forward networks have also been applied to the problem which has shown small, statically significant improvements in the prediction.

### Signature verification:

A signature verification system is based upon the similarity between signatures. The system is composed into two major components : (1) the signature preprocessing component which focuses on the timing information and positioning of the pen point while making signatures and (2) machining learning component for modeling the extracted features. Recurrent neural networks have been successfully applied for modeling signatures [148]. A training set of extracted features from signatures is used for training which may contain both positive and negative samples. Upon successful training, the network will be presented with new signatures of people who were included in training. The network thus has to predict whether a given signature belongs to a person. Signature verification systems can be used in banks, airports and security systems.

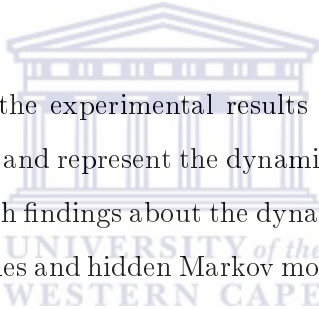
## 3.5 Summary

We have seen how the strengths of intelligent system paradigms can be combined into hybrid systems. We have discussed the combination of neural networks with symbolic knowledge in symbolic connectionist learning. We have seen how the strengths of neural networks and expert systems can be combined into a hybrid system which can explain how it arrived at a particular solution and also learns from past experience. Upon presentation of new data, the expert system can update its existing rules through learning in order to keep up-to-date with a changing environment. These systems have contributed to a wide range of applications including pattern recognition problems. Finally, we have shown and discussed in detail our proposed hybrid system. We have discussed the possible training methods of hybrid recurrent neural networks.

In the next chapter, we will discuss the implementation of hybrid HMM-RNN architecture on sample test beds such as automata theory to show that it can learn and represent the dynamical systems.

# Chapter 4

## Sample test beds and implementation results



In this chapter, we present the experimental results on sample test beds to show that the hybrid architecture can learn and represent the dynamical systems using finite state automata. Here, we discuss some research findings about the dynamical systems, automata theory, formal languages, finite state machines and hidden Markov models. Later on, the detailed explanation about the formal languages, automata theory, finite state machines related hidden Markov models is presented. Finally, we present the implementation and the results of the hybrid HMM-RNN algorithm on sample test beds with a concluding summary of this chapter.

### 4.1 Introduction

In this section, we present the relevant research findings on RNNs and HMMs related to the automata theory and dynamical systems, etc.

Finite-state automata represent dynamical behavior and are useful frameworks for studying recurrent neural networks as no feature extraction is necessary. A deterministic finite automaton is a finite automaton where for each pair of state and input signal, there is one transition to the next state. A deterministic finite automaton reads in a string of input symbols. For each input symbol, it performs a state transition. When the last input symbol has been received, the automaton will either accept or reject the string depending on the output

of the state. Hidden Markov models are finite state machines and have been successfully applied for modeling speech sequences.

Here, we present an overview of relevant research review on dynamical systems using RNN and theory of automation.

In [172], Alessando explained that recurrent neural networks can simulate any finite state automata as well as any multi-stack Turing machine. When constraining the network architecture, however, this computational power may no longer hold. For example, recurrent cascade-correlation cannot simulate any finite state automata. Thus, it is important to assess the computational power of a given network architecture, since this characterizes the class of functions which, in principle, can be computed by it. These authors discussed the computational power of neural networks for structures. Elman-style networks, cascade-correlation networks and neural trees for structures are introduced. Their validated results indicates that that Elman-style networks can simulate any frontier-to-root tree automation, while neither cascade-correlation networks nor neural trees can. The final results indicates that the neural trees for sequences cannot simulate any finite state machine.

Trentin and Cattoni [182] proposed an hybrid system for modeling, learning and recognition of sequences of states in indoor robot navigation. States are broadly defined as local relevant situations (in the real world) in which the robot happens to be during the navigation. The hybrid is based on parallel Recurrent Neural Networks trained to perform a posteriori state probability estimates of an underlying hidden Markov model given a sequence of sensory (e.g. sonar) observations. Their approach was suitable for navigation and for map learning. These authors explored the recognition of noisy sequences acquired by a mobile robot equipped with 16 sonars.

A connectionist model dynamical system proposed by Baldi *et al.* [12] for learning in sequential domains which uses hidden states to store contextual information. In principle, these models can adapt to variable time lags and perform complex sequential mappings. In spite of several successful applications (mostly based on hidden Markov models), the general class of sequence learning problems is still far from being satisfactorily solved. A dynamical system was said to be causal if the output at (discrete) time  $t$  does not depend on future inputs. Causality was easy to justify in dynamics that attempt to model the behavior of many physical systems. Clearly, in these cases the response at time  $t$  cannot depend on



stimulae that the system has not yet received as input. As it turns out, non-causal dynamics over infinite time horizons cannot be realized by any physical or computational device. For certain categories of finite sequences, however, information from both the past and the future can be very useful for analysis and predictions at time  $t$ . This is the case, for example, of DNA and protein sequences where the structure and function of a region in the sequence may strongly depend on events located both upstream and downstream of the region, sometimes at considerable distances.

In [101], Ivan and Andrej explained about the finite state automata, which can be treated as general discrete dynamical systems from the view of systems theory. These authors addressed the problem of unconditional on-line identification of an unknown finite automaton. They proposed a generalized architecture of recurrent neural networks with a corresponding on-line learning scheme as a solution to the problem. An on-line rule-extraction algorithm is further introduced. They presented the proposed architecture and tested on the on-line learning scheme and the on-line rule-extraction methods, strongly connected automata, ranging from a very simple example with two states only to a more interesting and complex one with 64 states; their results indicates that both training and extraction processes are very promising.

A method for combining the evolutionary hill climbing with incremental learning and a well-balanced training set presented was by Stephan *et al.* [37], which enables the first order recurrent networks to reliably learn context-free and mildly context-sensitive languages. These authors trained the networks to predict symbols in string sequences of the context-sensitive language. Comparative experiments with and without incremental learning indicated that incremental learning can accelerate and facilitate training. Furthermore, incrementally trained networks generally resulted in monotonic trajectories in hidden unit activation space, while the trajectories of non-incrementally trained networks were oscillating. They finally concluded that non-incrementally trained networks were more likely to generalize.

A recurrent neural network architecture given by Hiroyuki *et al.* [90] was capable of incremental learning and test the performance of the network. In incremental learning, the consistency between the existing internal representation and a new sequence is unknown, so it was not appropriate to overwrite the existing internal representation on each new sequence. Their proposed model states that the parallel pathways from input to output are preserved as

possible, and the pathway which has emitted the wrong output is inhibited by the previously fired pathway. Accordingly, the network begins to try other pathways ad hoc. This modeling approach was based on the concept of the parallel pathways from input to output, instead of the view of the brain as the integration of the state spaces. Their validation study indicates that the extension of this approach to building a model of the higher functions such as decision making.

A problem in the control of automata on infinite strings was defined and analyzed by Thistle and Wonham [176]. In order to investigate the development of a fixpoint characterization of the controllability subset of a deterministic rabin automaton, the set of states from which the automaton can be controlled to the satisfaction of its own acceptance condition. Their representation of fixpoint allows straightforward computation of the controllability subset and the construction of a suitable state-feedback control for the automaton. Their results indicates that the applications to control synthesis, automaton synthesis, and decision procedures for logical satisfiability; in particular, they represent a direct, efficient and natural solution to Church problem, the construction of winning strategies for two-player zero-sum  $\omega$ -regular games of perfect information, and the emptiness problem for automata on infinite trees.

A decentralized adaptive synchronization problem for a simple yet nontrivial discrete-time stochastic model of network dynamics was investigated by Bin *et al.* [22], which also illustrates a general framework for a class of adaptive control problems for complex systems with uncertainties. They describe synchronization phenomena in noisy environments, several new definitions of synchronization for stochastic systems are given and applied in our model. In the framework proposed and proved that in four different cases on local goals, including deterministic tracking, center-oriented tracking, loose tracking, and tight tracking, under mild conditions on noise sequence and communication limits, the agents in the considered model can achieve global synchronization in sense of mean by using local estimators and controllers based on a least-squares (LS) algorithm. Their results indicates that agents in a complex system disturbed by noise with communication limits can autonomously achieve the global goal of synchronization by using local LS-based adaptive controllers while they are pursuing for their local goals.

In [152], Razvan and Herbert explained a simple working memory model in which all of the performance modes are trained into a recurrent neural network (RNN) of the echo state

network (ESN) type. They demonstrated the model on a bracket level parsing task with a stream of rich and noisy graphical script input. In terms of nonlinear dynamics, memory states correspond, intuitively, to attractors in an input-driven system. These authors also contributed to a rigorous formal framework to describe such attractors, generalizing from the standard definition of attractors in autonomous (input-free) dynamical systems.

A model reduction technique for the class of discrete space and time hidden Markov models was presented by Wu and Noe [202]. In order to efficiently simulate and analyze large scale stochastic models, which are very relevant in the fields of computational biology, finance, social sciences, etc. Their method was illustrated on some model applications to preserve the dynamical properties of the system.

Torkestani [177] designed an automata-based focused web crawler. Taking advantage of learning automata, they proposed a crawler learns the most relevant URLs and the promising paths leading to the target on-topic documents. It can effectively adapt its configuration to the Web dynamics. This crawler is expected to have a higher precision rate because of construction a small Web graph of only on-topic documents. Based on the Martingale theorem, the convergence of the proposed algorithm is proved. To show the performance of the proposed crawler, extensive simulation experiments are conducted. Their indicated results show the superiority of the proposed crawler over several existing methods in terms of precision, recall, and running time. Their  $t$ -test is used to verify the statistical significance of the precision results of the proposed crawler.

In [168], Shibata and Yoshinaka presented several collapsed Bayesian methods, which work by first marginalizing out transition probabilities, for inferring several kinds of probabilistic finite automata. Their methods include collapsed Gibbs sampling (CGS) and collapsed variational Bayes, as well as two new methods. Their targets range over general probabilistic finite automata, hidden Markov models, probabilistic deterministic finite automata, and variable-length grams. They implemented and compared these algorithms over the data sets from the Probabilistic Automata Learning competition, which were generated by various types of automata. They reported that the CGS-based algorithm designed to target general probabilistic finite automata performed the best for any types of data.

Tracking problem for control systems using recurrent neural network was presented by Michael *et al.* [125] and they demonstrated that a major difficulty in training any RNN is

the problem of exploding gradients and proposed a solution to this in the case of tracking problems, by introducing a stabilization matrix and by using carefully constrained context units. Their solution allows to achieve consistently lower training errors and hence allows us to more easily introduce adaptive capabilities. The resulting RNN is one that has been trained off-line to be rapidly adaptive to changing plant conditions and changing tracking targets. The case study they used is a renewable-energy generator application; that of producing an efficient controller for a three-phase grid-connected converter. The controller they produced can cope with the random variation of system parameters and fluctuating grid voltages. It produces tracking control with almost instantaneous response to changing reference states, and virtually zero oscillation. This compares very favorably to the classical proportional integrator (PI) controllers, which we show produce a much slower response and settling time. Their RNN finally exhibits better learning stability and convergence properties and can exhibit faster adaptation, than has been achieved with adaptive critic designs.

Rest of the chapter is organized as follows: In the next section, we deal with the finite state automata and the knowledge representation using HMMs and the relation to the hybrid architectures has been discussed. Finally, the experimental results on the implementation of the gradient descent algorithm for hybrid HMMs-RNNs on automata theory, which is one of the sample test bed used for this study has been presented with the concluding summary.

## 4.2 Finite-state automata and knowledge representation

In this section, we present an overview of the formal languages and the finite state automata representation, deterministic finite state automata, finite state machines related to the hidden Markov model.

Symbolic or expert knowledge can be inserted into neural networks prior to training for better training and generalization performance. It has been shown that deterministic finite-state automata can be directly encoded into recurrent neural networks prior to training. Initially, neural networks were viewed as black boxes as they could not explain the knowledge learnt in the training process, i.e., it was believed to be difficult to understand the

knowledge or its representation in the weights as part of the information processing in the network. Knowledge extraction is the process of finding the meaning of the internal weight representation of the network.

For recurrent neural networks, knowledge can be extracted in the form of finite-state automata which gives insight into knowledge representation in recurrent neural networks. Recurrent neural networks have been trained on deterministic finite-state automata (DFA's) and knowledge extraction methods have been applied to explore the knowledge representation in the weights of the trained network.

### **Formal languages:**

Formal languages are useful for studying recurrent neural networks for a number of reasons;

- There is no need for feature extraction,
- They allow us to study the knowledge representation in recurrent neural networks and
- The dynamics of many real world processes can be represented as finite-state processes

at some level of abstraction [114]. Finite-state automata encoding, induction and extraction has been studied extensively [45, 66, 119, 191].

### **Finite-state automata:**

A finite-state automata is a device that can be in one of a finite number of states. Under certain conditions, it can switch to another state; this is called a transition. When the automaton starts processing input, it can be in one of its initial states. Some automata contain another important subset of states: the final (or accepting) states. If an automaton is in a final state after processing an input sequence, it is said to accept or reject its input according to the output membership of the last state. We use finite-state automata as test beds for training recurrent neural networks. Presumably, strings used for training do not need to undergo any feature extraction. They are used to show that recurrent neural networks can represent dynamical systems.

### Deterministic finite-state automata:

A deterministic finite-state automata (denoted by  $M$ ) is defined as

$$M = (Q, \Sigma, \delta, q_0, F),$$

where

- $Q$  is a finite set of states;
- $\Sigma$  is an input alphabet;
- $q_0 \in Q$  is the initial state;
- $F \subseteq Q$  is the set of final states; and
- $\delta$  maps  $Q \times \Sigma$  to  $Q$ .

$M$  has a finite control, an input tape, and a read head. In one move,  $M$ , in state  $q$ , reads input symbol  $a$ , changes state to  $\delta(q, a)$ , and moves the read head one symbol to the right. If  $M$  is in an accepting state when the read head moves off the end of the tape, then  $M$  *accepts* the input.

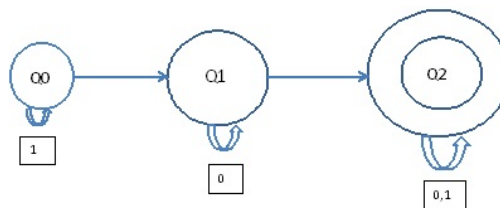


Figure 4.2.1: Example of a deterministic finite state automata

In Figure 4.2.1, we present an example of deterministic finite-state automata. Double circles show accepting states. Rejecting states are shown by single circles.

The DFAs are widely used in text editors for pattern matching, in compilers for lexical analysis, in web browsers for html parsing, and in operating systems for graphical user interfaces. They also serve as the control unit in many physical systems including: vending

machines, elevators, automatic traffic signals, and computer microprocessors. Also network protocol stacks and old VCR clocks. They also play a key role in natural language processing and machine learning.

A DFA captures the basic elements of an abstract machine: it reads in a string, and depending on the input and the way the machine was designed, it outputs true or false. A DFA is always is one of  $N$  states, which we name 0 through  $N - 1$ . Each state is labeled true or false. The DFA begins in a distinguished state called the start state. As the input characters are read in one at a time, the DFA changes from one state to another in a pre-specified way. The new state is completely determined by the current state and the character just read in. When the input is exhausted, the DFA outputs true or false according to the label of the state it is currently in.

### Finite state machines and hidden Markov models:

Since during 1990's to till date, finite state automata (FSA) and hidden Markov models (HMMs), have been used quite successfully to address several complex sequential pattern recognition problems, such as speech recognition, handwritten recognition, time series prediction, biological sequence analysis, and many others.

Finite State Automata allows complex learning problems to be solved by assuming that the sequential pattern can be decomposed into piecewise stationary segments, encoded through the topology of the FSA. Each stationary segment can be parametrized in terms of a deterministic or stochastic function. In the latter case, it may also be possible that the SFSA state sequence is not observed directly but is a probabilistic function of the underlying finite state Markov chain. This thus yields to the definition of the powerful HMMs, involving two concurrent stochastic processes: the sequence of HMM states modeling the sequential structure of the data, and a set of state output processes modeling the (local) stationary character of the data.

Let us assume that  $f$  be the transition function and  $g$  is the emission function. Any automation is called a stochastic, when the transition and emission functions are probabilistic. Regular Markov model, can be called as a stochastic automata when the the functions in the model are probabilistic and deterministic.

From the view of automation theory, an HMM differs basically from two features, mainly,

when the state of the model is not observable and if it contains the probabilistic emission function.

Informally speaking, an HMM is a variant of a finite state machine. However, unlike finite state machines, they are not deterministic. A normal finite state machine emits a deterministic symbol in a given state. Further, it then deterministically transitions to another state. HMMs do neither deterministically, rather they both transition and emit under a probabilistic model.

Usually, with a finite state machine, a string of symbols can be given and it can be easily determined

1. if the string could have been generated by the finite state machine in the first place and
2. the sequence of state transitions it was undertaken.

From the view of automata theory, an HMM diverge basically from two features, mainly, the state of the model is not observable and also if it has a probabilistic emission function. With an HMM, (1) is replaced with a probability that the HMM generated the string and (2) is replaced with nothing.

In general, the exact sequence of state transitions undertaken is hidden, hence the name. A Markov model is a probabilistic process over a finite set,  $S_1, \dots, S_k$ , usually called its states and each state-transition generates a character from the alphabet of the process.

In computing such processes, if they are reasonably complex and interesting, are usually called Probabilistic Finite State Automata (PFSA) or Probabilistic Finite State Machines (PFSM) because of their close links to deterministic and non-deterministic finite state automata as used in formal language theory.

In a regular Markov model, the state is directly visible to the observer. Therefore, the state transition probabilities are the only parameters. In an HMM, the state is not directly visible; however, the variables influenced by the states are visible. Each state has a probability distribution over the possible output tokens. The sequence of tokens generated by a HMM gives some information about the sequence of states. In a first order HMM, the state at time  $t + 1$  depends only on state at time  $t$ , regardless of the states in the previous times.

The term hidden hints at the process state transition sequence which is hidden from the observer. The process reveals itself to the observer only through the generated observable signal. A HMM is parameterized through a matrix of transition probabilities between states



and output probability distributions for observed signal frames given the internal process state. Thus, other than a deterministic finite state automata, where the state sequence, given an input sequence, is uniquely determined, a HMM gives only a probability measure for a state sequence matching a given input sequence.

We assume that for every time step  $t$  the system is in state  $w(t)$  and emits some visible symbol  $v(t)$ . We define a particular sequence of visible states as and thus we have . The model is the that in any state we have a probability of emitting a particular visible state  $v^{(t)} = \{v(1), v(2), \dots, v(T)\}$ . We denote this probability  $P = b_{jk}$ . Since we only have access to the visible states, while the are unobservable. The model is called an HMM as shown in the Figure 4.2.2.

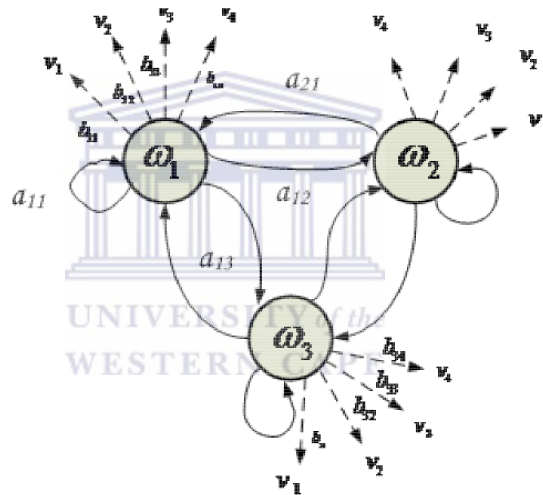


Figure 4.2.2: Hidden Markov model with emitting states

In Figure 4.2.2, we present an hidden Markov model. It represents three hidden units in a HMM and transitions between them are shown in solid lines. The visible states and the emission probabilities of visible states are shown with dashed lines. This model shows that all transitions are possible. In other HMMs, some candidate transitions are not allowed.

The following are some drawbacks of the HMM proposed by [28]:

- The training algorithm which maximizes the likelihoods instead of posteriori probabilities is mainly due to the poor discrimination.
- A priori choice of model topology and statistical distributions, e.g., assuming that

the probability density functions associated with the HMM state can be described as multivariate gaussian densities, each with a diagonal-only covariance matrix.

- First-order Markov chains assumption is the state sequences.
- Assumption that the input observations are not correlated over time. Thus, apart through the HMM topology, the possible temporal correlation across features associated with the same HMM state is simply disregarded.

In order to overcome some of these problems, many researchers have concentrated on integrating Artificial Neural Networks into the HMMs.

### **Stochastic finite state automata based on RNN:**

The idea of combining HMMs and RNNs was motivated by the observation that HMMs and RNNs had complementary properties:

- HMMs are clearly dynamic and very well suited to sequential data, but several assumptions limit their generality,
- RNNs can approximate any kind of nonlinear discriminant functions, are very flexible and do not need strong assumptions about the distribution of the input data,

but they cannot properly handle time sequences (although recurrent neural networks can indeed handle time, they are known to be difficult to train long term dependencies, and cannot easily incorporate knowledge in their structure as it is the case for HMMs).

### **Hybrid HMM-RNN Systems in terms of formalism:**

The HMMs are based on a strict probabilistic formalism, making them difficult to interface with other modules in a heterogeneous system.

When using these posterior probabilities (instead of local likelihoods) in stochastic FSA, the model becomes a recognition model, where the observation sequence is an input to the system, and where all local and global measures are based on a posteriori probabilities.

These hybrid HMM-RNN approaches provide more discriminant estimates of the emission probabilities needed for HMMs, without requiring strong hypotheses about the statistical distribution of the data.

In the next section, we present the implementation of developed learning algorithm of hybrid HMM-RNN architecture on sample test beds and finally we concluded with the summary followed by a brief idea of the next chapter.

### 4.3 Experimental results

In the current study, we investigate to show how finite automaton can be used to train recurrent neural networks making them suitable for modeling dynamical systems. We will train first-order recurrent neural networks on deterministic finite-state automata to show their knowledge acquisition. Finally, we will show how our proposed hybrid recurrent neural networks architecture based on hidden Markov models can train and represent finite automaton making them suitable for modeling dynamical systems.

#### **Recurrent neural networks as models of computation:**

Finite state automata and their corresponding languages can be viewed as a general paradigm of temporal, symbolic language. There is no feature extraction necessary for recurrent neural networks to learn these languages. The knowledge acquired in recurrent neural networks through learning well corresponds with the dynamics of finite-state automata. The representation of automata as a prerequisite for learning its corresponding languages, i.e., if the architecture cannot represent a particular automaton then it would not be able to learn it either. It has been shown that recurrent networks can represent certain automaton and automaton can be mapped directly into recurrent networks.

Recurrent neural networks are systems that model dynamical processes. Formal languages such as finite state automata have characteristics of dynamical systems. Using finite state automata we can show that recurrent neural networks can learn and represent dynamical systems. Recurrent neural networks can be trained with strings whose labels are assigned by deterministic finite state automata.

We generate the training data set by presenting string length of 1 to 10 to corresponding finite automaton which labels the output with each corresponding string. Similarly, we generate a testing data set for string lengths from 1 to length 15. In time varying sequences, longer patterns represent long time dependencies. Gradient descent has difficulties in learning long time dependencies as error gradient vanishes with increasing duration of dependencies. Incremental data learning addresses this problem by learning through working sets of the complete training set patterns of increasing lengths. In this way, short time dependencies are learnt first which then helps the network to learn longer time dependencies. A working set contains patterns of increasing order of lengths. The networks trains on each working set of a number of training epochs until the network converges. The training is terminated when the network performs satisfactorily on the entire training set or iterates through all working sets of the training set.

In Figure 4.3.1, we present a ten state deterministic finite state automata which is used for training the hybrid recurrent network architecture interpreted by hidden Markov models.

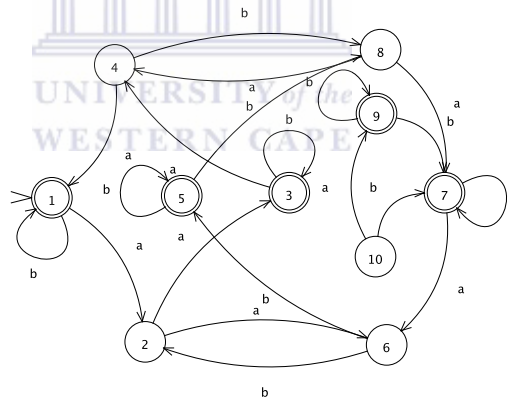


Figure 4.3.1: Ten state deterministic finite state automata

An example of 10 State deterministic finite state automata is presented in Figure 4.3.1. Double circles show accepting states. Rejecting states are shown by single circles while state 1 is the automaton start state. The training and testing set is obtained upon presentation of strings to this automaton which gives an output, i.e., a rejecting or accepting state depending on the state where the last sequence of the string was presented. For example, the output of a string of length 10, i.e., aaaabaaaba in alphabet  $\{0,1\}$  is state 7. It is an accepting state, therefore the output is 1.

### **Deterministic finite-state automata training:**

To show the effectiveness and efficiency of the proposed algorithm of the hybrid architecture, the following tests were performed.

We generate the training data set by presenting string length of 1 to 10 to a deterministic finite automaton. The deterministic finite automaton labels the output on each string it takes as input depending on the state where the final symbol of the string was presented. Similarly, we generate a testing data set for string lengths from 1 to length 15. We obtain a training data set with 2048 string samples and a testing data set of 65535 strings.

We used the first order recurrent neural networks architecture with the following topology: 2 neurons in the input layer which represents the string input and 1 neuron in the output layer representing the string output. We used a learning rate value of 0.3, momentum rate value of 0.9, sigmoid sensitivity rate of 11.5 and ran experiment with 5, 10 and 15 neurons in the hidden layer along with the hidden Markov model Parameters (No. of observations = 10, No. of observation symbols = 2, No. of States = 3).

While deriving the equations for Hybrid HMM-RNN architecture, we finally ended up with the initial values, weights and observation probabilities of the network. We initially started with a training cycle which contained 2 and 3 samples in a working set of increasing string lengths and ran the experiment up to 50 cycles and started training again with the same set of weights that were obtained from training in the previous cycle and continued till 100 cycles. The network trains on each working set on a number of training cycles until it converges to a solution. The training is terminated when the network performs satisfactorily on the entire training set or iterates through all working sets of the training set. We calculated the training and generalization performances of the data ie., it has correctly classified/learned all the strings in the training and testing sets. The network is presented with data in the testing set and the performance of the networks is determined upon its generalization on the test data. The Table 4.3.1 shows the HMM parameters which are used for processing the hybrid HMM-RNN architecture, and tables 4.3.2 and 4.3.3 show the results obtained for single order RNN and Hybrid HMM-RNN. Here, we carried out two experiments

- Case 1: Simple DFA learning using RNN, and
- Case 2: Hybrid HMM-RNN architecture implementation of DFA.

In the first case, the training and generalization performances were quite good and achieved an average of 85 percent performance using simple RNNs, i.e., It has been shown that it correctly classifies all the strings in the training set which means the total number of accepted and rejected strings in the training and the testing sets and in the second case i.e., using the Hybrid HMM-RNN architecture, initially we got zero performance with -1 to 1 and -3 to 3 weight ranges and when we increase the number of hidden units from 5, 10, 15 and weight ranges between -5 to 5, -7 to 7 we got an average of 83 percent on the performances.

### Learning Deterministic Finite Automaton:

The tables 4.3.2 and 4.3.3 shows the training and generalization performances of single order RNN and hybrid HMM-RNN architecture with different number of neurons in the input and hidden layers, weight ranges and number of training cycles for learning deterministic finite state automaton.

Table 4.3.1: Hidden Markov model parameters

Number of States	Number of Observations	Number of Observation Symbols
3	10	2

Table 4.3.2: Single order recurrent neural network

Input Neurons	Hidden Neurons	Training Cycles	Training MSE	Testing MSE	Generalization MSE
3	5	50	0.194652	92.9169	87.9641
3	5	101	0.177034	78.9566	70.6649
3	5	500	0.150888	85.9326	82.0357
5	10	1000	0.0822267	83.0292	81.4205
5	10	1500	0.0886557	72.4865	71.4137
5	10	2000	0.0854211	60.1392	50.3495
10	15	2500	0.0787231	79.2631	74.1746
10	15	5000	0.184652	68.5339	64.9201

Finally, in order to reveal that the hybrid recurrent neural network has good recognition/classification and generalization performance to solve the first order and hybrid HMM-RNN problems, the results show that recurrent neural networks can learn deterministic finite

Table 4.3.3: Learning deterministic finite state automation using hybrid HMM-RNN

Hidden Neurons	Weight Range	Training Cycles	Training Performance	Generalization Performance
1	-1 to +1	100	0%	0%
3	-3 to +3	100	0%	0%
5	-5 to +5	53	84.5%	79.3%
10	-7 to +7	31	82.2%	81.3%
15	-15 to +15	9	0%	0%

state automata by means of gradient descent learning. They show good training and generalization performance compared to other models and it is also seen that the number of training epochs in the last cycle vary for different neural networks topologies depending on the number of neurons in the hidden layer.

## 4.4 Summary

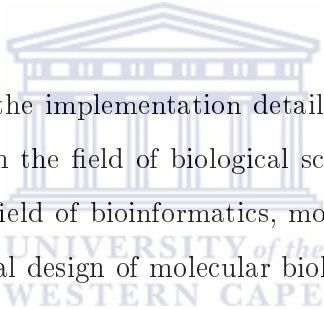
In this chapter, we have discussed finite automata which have been the basis of studying knowledge representation of recurrent neural networks. Finite State Automata can be characterized as deterministic, fuzzy or probabilistic automata.

The constructed and proposed hybrid HMM-RNN architecture shown that they can train and represent dynamical systems such as deterministic finite automaton. Our results in general show that finite automaton can be trained and represented by recurrent neural networks and hybrid recurrent networks based on hidden Markov models.

In the next chapter, we present the implementation of hybrid HMM-RNN algorithm on a real world application like Enzyme Classification.

# Chapter 5

## Real time application - Enzyme classification



In this chapter, we present the implementation details of the hybrid HMM-RNN algorithm for a real time application in the field of biological sciences. The relevant research findings have been presented in the field of bioinformatics, molecular biology and genomics; we give an overview about the central design of molecular biology, protein classification and enzyme classification. Finally, we present the results of enzyme classification using our hybrid HMM-RNN architecture.

### 5.1 Introduction

Generally, the biological applications consist of huge amount of data to be processed. These data are analyzed to improve the performance of the existing systems using different types of algorithms that are based on the classical theory of hidden Markov models. In this thesis, we are interested to apply hybrid HMM-RNN algorithm. Before we proceed further, we wish to emphasize that the hybrid HMM-RNN architecture can be applied to the following biological applications:

- Biological sequence analysis,
- Prediction of protein structures,



- Clustering of sequences,
- Pattern classifications,
- Enzyme classifications, etc.

The Hybrid HMM-RNN algorithm can also be used in the following real time environments, i.e., Bioinformatics, Biometrics, Speech recognition, Time series analysis, Robotics, Data mining, etc..

As far as the biological applications are concerned, during the course of our study, we have explored the use of hybrid HMM-RNN architecture to model the proteins. We have compared the performances with a single HMM architecture with and without using a prior knowledge. Those results are partial and we are still working on optimizing them.

Another application of our hybrid architecture that we have explored is a real time protein sequence analysis, namely, enzyme classification in bioinformatics. We obtained successful results which we present in detail in this chapter.

Before we proceed further, we present a recent research findings on enzyme classification in the field of ANNs, RNNs, HMMs that are relevant to this study.

A model developed for predicting the function of a protein from its amino-acid sequence was presented by Des *et al.* [52]. Given features that can be computed from the amino-acid sequence in a straightforward fashion (such as pI, molecular weight, and amino-acid composition), the technique allows us to answer questions such as: Is the protein an enzyme? If so, in which Enzyme Commission (EC) class does it belong? Their proposed approach uses machine learning (ML) techniques to induce classifiers that predict the EC class of an enzyme from features extracted from its primary sequence. These authors explored the following experiments with the use of three different ML techniques in conjunction with training datasets derived from PDB and from Swiss-Prot. They also demonstrated the use of several different feature sets. Their method predicts the first EC number of an enzyme with 74% accuracy (thereby assigning the enzyme to one of six broad categories of enzyme function), and to predict the second EC number of an enzyme with 68% accuracy (thereby assigning the enzyme to one of 57 subcategories of enzyme function). Their validation results indicates that the proposed technique could be a valuable complement to sequence-similarity searches and to pathway-analysis methods.

A method to predict an enzyme class that combines the strengths of statistical and data mining methods was presented by Luiz *et al.* [27]. Predicting enzyme class from protein structure parameters is a challenging problem in protein analysis. Their proposed method has a strong mathematical foundation and is simple to implement, achieving an accuracy of 45%. Their validation results demonstrated an excellent performance after comparing with the methods found in the literature designed to predict enzyme classes indicates that their method outperforms the existing methods.

The hybrid model proposed and tested by Osman *et al.* [137] applied to different topologies of network architecture, especially in determining the number of hidden nodes. These authors indicated that the proposed model results are quite promising in classifying the enzymes from the nucleic acid and protein sequences which are taken from protein data bank. Protein sequence classification deals with the assignment of sequences to known categories based on homology detection properties developed using multilayer perceptron with the genetic algorithm and backpropagation. Their research results demonstrated that the proposed hybrid model which was tested with different topologies of network architecture, especially in determining the number of hidden nodes are quite promising in classifying the enzyme accordingly.

Ali and Shawky [3] built a new hybrid model for protein classification using Fourier transform method. Proteins that are evolutionarily and thereby functionally related are said to belong to the same classification. Identifying protein classification is of fundamental importance to document the diversity of the known protein universe. It also provides a means to determine the functional roles of newly discovered protein sequences. Their goal is to predict the functional classification of novel protein sequences based on a set of features extracted from each protein sequence and they proposed the technique that used datasets extracted from the structural classification of proteins (SCOP) database. A set of spectral domain features based on Fast Fourier Transform (FFT) was used. These authors proposed a classifier which uses multi-layer back propagation (MLBP) neural network for protein classification. The maximum classification accuracy is about 91% when applying the classifier to the full four levels of the SCOP database. However, it reaches a maximum of 96% when limiting the classification to the family level. Their research results reveals that the classification results that spectral domain contains information that can be used for classification with high

accuracy. In addition, the results emphasize that sequence similarity measures are of great importance especially at the family level.

Automatic classification of proteins using machine learning was given by Zimek *et al.* [208] is an important problem that has received significant attention in the literature. One feature of this problem is that expert-defined hierarchies of protein classes exist and can potentially be exploited to improve classification performance. They investigated empirically whether this is the case for two such hierarchies and they compare multi-class classification techniques that exploit the information in those class hierarchies and those that do not, using logistic regression, decision trees, bagged decision trees, and support vector machines as the underlying base learners. In particular, they compare hierarchical and flat variants of ensembles of nested dichotomies. The latter have been shown to deliver strong classification performance in multi-class settings. They demonstrated the results for synthetic, fold recognition, enzyme classification, and remote homology detection data. Their results indicates that exploiting the class hierarchy improves performance on the synthetic data, but not in the case of the protein classification problems. The strong flat multi-class methods be used as a baseline to establish the benefit of exploiting class hierarchies in this area.

A methodology based on Artificial Neural Networks for protein functional classification was given by Thiago *et al.* [51]. A new protein coding scheme, called here Extended-Sequence coding by Sliding Windows, is presented with the goal of overcoming some of the difficulties of the well method sequence coding by Sliding Window. The new protein coding scheme uses more than one sliding window length with a weight factor that is proportional to the window length, avoiding the ambiguity problem without ignoring the identity of small subsequences. Accuracy for Sequence Coding by Sliding Windows ranged from 60.1 to 77.7 percent for the first bacterium protein set and from 61.9 to 76.7 percent for the second one, whereas the accuracy for the proposed Extended-Sequence Coding by Sliding Windows scheme ranged from 70.7 to 97.1 percent for the first bacterium protein set and from 61.1 to 93.3 percent for the second one. Additionally, they also classified the protein sequences inconsistently by the Artificial Neural Networks that were analyzed by CD-Search revealing that there are some disagreement in public repositories, calling the attention for the relevant issue of error propagation in annotated databases due the incorrect transferred annotations.

In [128], Mohammed and Guda proposed an enzyme classes using different computational methods and can be used to annotate the enormous amount of unannotated enzyme sequences. For function prediction and classification of enzymes, features based on amino acid composition, sequence and structural properties, domain composition and specific peptide information have been widely used by different computational approaches. The total predicted accuracy in each feature space has its own merits and limitation. Prediction accuracy improves when machine-learning methods are used to classify enzymes. Given the incomplete and unbalanced nature of annotations in biological databases, ensemble methods or methods that bank on a combination of orthogonal feature are more desirable for achieving higher accuracy and coverage in enzyme classification.

The comparison of enzyme mechanistic descriptors derived from the MACiE (Mechanism, Annotation and Classification in Enzymes) database and use multivariate statistical analysis for assessment of enzyme classification was proposed by Neetika *et al.* [135]. As the volume of available information is increasing, a large number of informatics groups have tried to use protein sequence and structural information to understand and reproduce the classifications. They developed a computational protocol using the R package CARET to predict EC number from MACiE-derived descriptors. They evaluated 260 well annotated chemical reaction mechanisms of enzymes using machine learning methods, placing them into the six top level EC classes. Finally, they compared the classification performances of three supervised learning techniques, Support Vector Machine, Random Forest, and K-Nearest, for the reaction mechanism classification task using five different descriptor sets from MACiE data. Results: They finally found that all classifiers performed similarly in terms of overall accuracy with the exception of K-Nearest Neighbour analysis, which has the lowest performance. The best performance was achieved by the Random Forest classifier.

A method that distinguishes protein enzyme sequences from those of non-enzymes was given by Chetan *et al.* [40]. These authors presented the approaches that cluster enzymes based on their sequence and structural similarity have been identified. But, these approaches are known to fail for proteins that perform the same function and are dissimilar in their sequence and structure. A supervised machine learning model to predict the function class and sub-class of enzymes based on a set of 73 sequence-derived features has been modelled. The functional classes are as defined by International Union of Biochemistry and Molecular

Biology. Using an efficient data mining algorithm called random forest, They constructed a top-down three layer model where the top layer classifies a query protein sequence as an enzyme or non-enzyme, the second layer predicts the main function class and bottom layer further predicts the sub-function class. Their model reported overall classification accuracy of 94.87% for the first level, 87.7% for the second, and 84.25% for the bottom level. Their valid research results compare very well with existing methods, and in many cases report better performance. Using feature selection methods, they have shown the biological relevance of a few of the top rank attributes.

A novel ab initio predictor of protein enzymatic class was presented by Viola *et al.* [187]. This predictor can classify proteins, solely based on their sequences, into one of six classes extracted from the enzyme commission (EC) classification scheme and is trained on a large, curated database of over 6,000 non-redundant proteins assembled in this work. These authors developed “The predictor” which is powered by an ensemble of N-to-1 Neural Network, a novel architecture which N-to-1 Neural Networks operate on the full sequence and not on predefined features. All motifs of a predefined length (31 residues in this work) are considered and are compressed by an N-to-1 Neural Network into a feature vector which is automatically determined during training. Their validated results indicates that the predictor in 10-fold cross-validation and obtain state of the art results, with a 96% correct classification and 86% generalized correlation. All six classes are predicted with a specificity of at least 80% and false positive rates never exceeding 7%. They are still investigating the enhanced input encoding schemes which include structural information, and are analyzing trained networks to mine motifs that are most informative for the prediction.

The rest of this chapter is organized as follows. In Section 5.2 we present the design of molecular biology, hidden Markov models and neural networks in the field of bioinformatics. Section 5.3 deals with the application of protein and enzyme classifications. The results and relevant discussions are provided in Section 5.4. Finally, in Section 5.5, we provide a brief summary of the work presented in this chapter.

## 5.2 Central design of Molecular Biology

In this section, we present an overview of molecular biology, in particular, about its components, such as DNA, RNA, proteins, profile HMMs, protein classification, etc..

Proteins are composed of amino acids connected by peptide bonds. The primary structure of proteins is regarded as the linear sequence of amino acids in a polypeptide chain. Proteins can be grouped into families and these families into superfamilies according to features such as hydrophobicity, composition, structure, length, three-dimensional shape, and electric charge (eg isoelectric point) [116] with the objective of establishing the common biological functions.

The amino acid sequence of a protein ultimately determines its function. Proteins usually have segments in their sequence of amino acids known as motifs [9] that are crucial for their biological functions, and that can be used for their identification. There are 20 different types of amino acids presented in Table 5.3.1 that are combined in a linear sequence, which has the necessary information to generate a unique three-dimensional structure. Theoretically, the number of possible combinations of amino acids is infinite [116]. Amino acids, in turn, vary in the side chain. The physical and chemical properties of the side chains of the amino acids of a protein (for instance, the fact that some of them have affinity with water) are important for the folding of the protein and its function. Like the proteins that make, amino acids can be classified in several ways, such as by electric charge, molecular weight and hydrophobicity. Kyte and Doolittle [111] proposed a hydrophobicity scale for all 20 amino acids ranging from 4.0 (most hydrophilic) to +4.5 (most hydrophobic). This scale is shown in Table 5.3.1, where it can be seen that amino acids can be also categorized as hydrophobic, neutral or hydrophilic.

### DNA - Deoxyribonucleic acid:

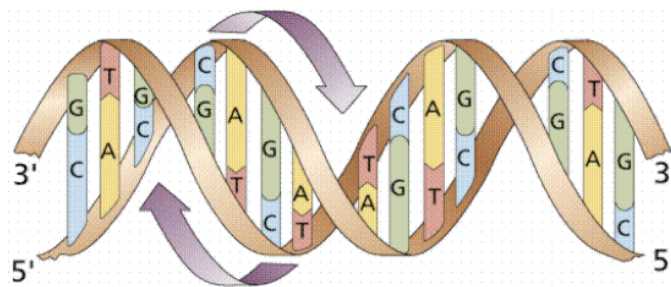


Figure 5.2.1: Example of a typical DNA molecule

In humans, as in other higher organisms, a DNA molecule consists of two strands that wrap around each other to resemble a twisted ladder whose sides, made of sugar and phosphate molecules, are connected by rungs of nitrogen containing chemicals called bases. Four different bases are present in DNA: adenine (A), thymine (T), cytosine (C), and guanine (G). The particular order of the bases arranged along the sugar- phosphate backbone is called the DNA sequence; the sequence specifies the exact genetic instructions required to create a particular organism with its own unique traits. The two DNA strands are held together by weak bonds between the bases on each strand, forming base pairs (bp). Genome size is usually stated as the total number of base-pairs; the human genome contains roughly 3 billion base-pairs. A gene is a segment of a DNA molecule (ranging from fewer than 1 thousand bases to several million), located in a particular position on a specific chromosome, whose base sequence contains the information necessary for protein synthesis.

### **RNA:**

RNA has the same structure as DNA. The primary differences between RNA and DNA are: RNA has a hydroxyl group on the second carbon of the sugar and instead of using nucleotide thymine, RNA uses another nucleotide called uracil (U). Since RNA has extra hydroxyl group on it's sugar strand, RNA is too bulky to form a stable double helix therefore it exists as a single-stranded molecule. In addition to that, because the RNA molecule is not restricted to a rigid double helix, it can form many different structures. There are several different kinds of RNA made by the cell. They are mRNA, tRNA, rRNA and snRNA.

### **PROTEIN's:**

Proteins are involved in almost all biological activities, structural or enzymatic. A protein is made by arranging amino acids together in a specific sequence (the sequence of every protein is different). These amino acids are held together by a special bond called a peptide bond. There are altogether 20 different amino acids.

How does the sequence of a strand of DNA correspond to the amino acid sequence of a protein? This concept is explained by the central theme of molecular biology, according to which:

- The DNA replicates its information in a process called replication that involves many enzymes.
- The DNA codes for the production of messenger RNA (mRNA) during transcription. In eukaryotic cells, the mRNA is processed (essentially by splicing) and migrates from the nucleus to the cytoplasm.
- Messenger RNA carries coded information to ribosomes. The ribosomes "read" this information and use it for protein synthesis. This process is called translation

The above can be viewed better through Figure 5.2.2.

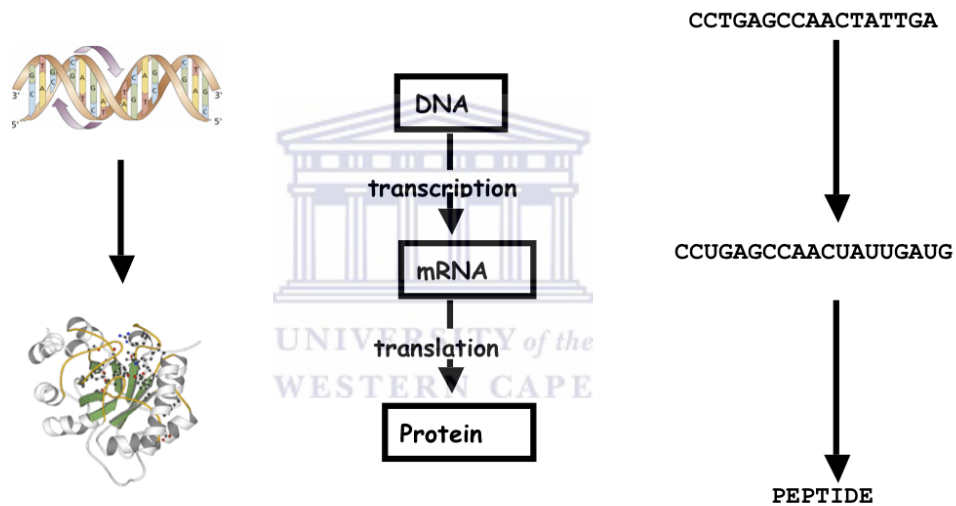


Figure 5.2.2: Hybrid HMM-RNN architecture

## Hidden Markov model's in bioinformatics:

Hidden Markov model contains the following:

- $\Pi$ -vector : contains the probability of the hidden model being in a particular hidden state at time  $t = 1$ ;
- state transition matrix : holding the probability of a hidden state given the previous hidden state;



- output matrix : containing the probability of observing a particular observable state given that the hidden model is in a particular hidden state.

Thus a hidden Markov model is a standard Markov process augmented by a set of observable states, and some probabilistic relations between them and the hidden states.

### Profile HMMs:

Protein structural similarities make it possible to create a statistical model of a protein family which is called a profile. The idea is, given a single amino acid target sequence of unknown structure, we want to infer the structure of the resulting protein. The profile HMM is built by analyzing the distribution of amino-acids in a training set of related proteins. This HMM in a natural way can model positional dependent gap penalties.

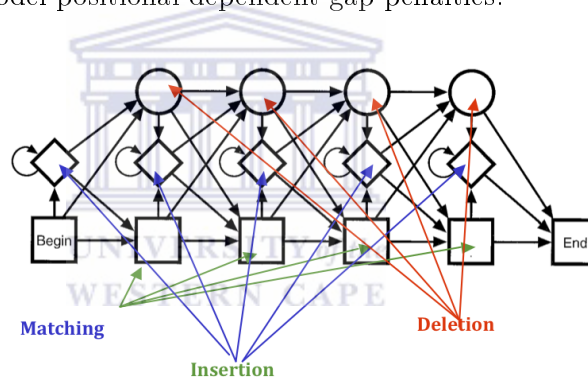


Figure 5.2.3: Topology of a profile HMMs

The basic topology of a profile HMM is shown in Figure 5.2.3. Each position, or module, in the model has three states. A state shown as a rectangular box is a match state that models the distribution of letters in the corresponding column of an alignment. A state shown by a diamond-shaped box models insertions of random letters between two alignment positions, and a state shown by a circle models a deletion, corresponding to a gap in an alignment. States of neighboring positions are connected, as shown by lines. For each of these lines there is an associated ‘transition probability’, which is the probability of going from one state to the other.

The match state represents a consensus amino acid for this position in the protein family. The delete state is a non-emitting state, and represents skipping this consensus position in the

multiple alignment. Finally, the insert state models the insertion of any number of residues after this consensus position.

A repository of protein profile HMMs can be found in the PFAM Database (see, e.g., <http://pfam.wustl.edu>). Building profiles from a family of proteins (or DNA) a profile HMM can be made for searching a database for other members of the family. Profile HMMs can also be used for the following:

- Scoring a sequence: We are calculating the probability of a sequence given a profile by simply multiplying emission and transition probabilities along the path.
- Classifying sequences in a database: Given a HMM for a protein family and some unknown sequences, we are trying to find a path through the model where the new sequence fits in or we are trying to align the sequence to the model. Alignment to the model is an assignment of states to each residue in the sequence. There are many such alignments and the Viterbi algorithm is used to give the probability of the sequence for that alignment.
- Creating multiple sequence alignment: HMMs can be used to automatically create a multiple alignment from a group of unaligned sequences. By taking a close look at the alignment, we can see the history of evolution. One great advantage of HMMs is that they can be estimated from sequences, without having to align the sequences first. The sequences used to estimate or train the model are called the training sequences, and any reserved sequences used to evaluate the model are called the test sequences. The model estimation is done with the forward-backward algorithm, also known as the Baum-Welch algorithm. It is an iterative algorithm that maximizes the likelihood of the training sequences.

### **Protein Classification:**

Protein sequence classification is one of the challenging and crucial problems of computational biology. The problem is to determine whether or not an unknown protein sequence belongs to a known set of class or family. If a new protein sequence belongs to a given class, it is presumed that it shares similar functions and structural characteristics. In several studies,

protein classification problem has been examined at various levels, according to a top hierarchy in molecular taxonomy, consisting of superfamilies, families and subfamilies [9].

Although, as yet, there is not consensus about protein classification, several properties can be used for this purpose, such as composition, number of side chains, 3-D folding shapes or biological function [116]. Another classification of protein database for the investigation of sequences and structures was proposed, in which the Murzin *et al.* [133] presented a model based on the protein domain. Many computational techniques have been used to classify proteins into families, such as structural transformations [136], data compression [41], genetic programming [108] and Markov chains [57]. They have demonstrated limited applicability and results. However, few studies published in the recent literature have applied Neural Networks to protein classification. This approach has gained some attention in the analysis of molecular sequences. For instance, in Wang *et al.* [190] proposed a new technique to extract data from proteins with a Bayesian Neural Network for classification. Wu *et al.* [201] explored the informative segments of sequences and used a three-layer Neural Network with a backpropagation algorithm for classification.

There are several approaches have been developed for solving protein classification. Most of them are based on appropriately modeling proteins families, either directly or indirectly [41]. Direct modeling technique means that by using a set of sequences namely training pattern, a model that characterizes the family of interest is built. The tool named HMMER using HMM employs a machine learning algorithm based on probabilistic graphical models to describe time-series and sequence data [57]. HMM is a generalization of the position-specific scoring matrix to include insertion and deletion states. HMM aligns an unknown sequence, to a given family if the scoring point is more significant than a cut-off value. Indirectly the techniques use a preprocessing tool to extract significant feature from sequences. In this way, sequences of variable length are transformed into fixed-length input vectors that are subsequently used for training discriminative models, such as neural networks [57].

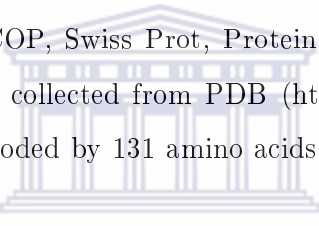
Enzymes are proteins that catalyze (i.e., accelerate) chemical reactions [134]. There are generally globular proteins which play a big role in determining the next steps that will occur in metabolic pathway inside living organisms. Without enzymes, metabolism would neither progress through the same steps, nor be fast enough to serve the needs of the cell.

## Protein Structure:

A striking characteristic of proteins is that they have very well defined 3-D structures. A stretched-out polypeptide chain has no biological activity, and protein function arises from the conformation of the protein, which is the 3-D arrangement or shape of the molecules in the protein. The native conformation of a protein is determined by a number of factors, and the most important are the four levels of structure found in proteins. Primary, secondary and tertiary refer to the molecules in a single polypeptide chain, and the fourth (quaternary) refers to the interaction of several polypeptide chains to form a multi-chained protein.

## Sequence encoding:

There are many protein databases available in open source websites, such as Protein Information Resources (PIR), SCOP, Swiss Prot, Protein Data Bank (PDB). In our work, the training-testing datasets are collected from PDB (<http://www.rcsb.org/pdb>). The Figure 5.2.4 represents an enzyme coded by 131 amino acids named Steroid Delta-isomerase which belongs to Isomerases family.



```
MNLPTAQEVQGLMARFIELVDVGDIEAIVQMYADDATVEDPFGQPPHGREQIAAFFRQG
LGGGKVRACLTGPPVRASHNGCGAMPFRVEMVWNGQPCALDVIDVMRFDEHGRIQTMQA
YWSEVNLSVREPQ
```

Figure 5.2.4: Tested sequence - Isomerase

For *protein classification*, two types of extracting feature from sequence are conducted: one is related to the global structure, and the other is related to the local similarity of sequence. Global feature is usually made by using 2-gram encoding scheme that count occurrence of two consecutive amino acids in protein sequence [26].

*Enzymes* are composed by a variable number of amino acids. We focus on encoding directly primary structure of protein, in string of letters forms into a numerical vector that appropriate for RNNs. The main idea of encoding procedure is by using Kyte and Doolittle hydrophobicity scale as seen in Table 5.3.1 to convert a string of amino acids symbol into real-valued vector.

RNNs are a neurobiological inspired systems that emulates the functioning of the brain, based on the way we believe that neurons work, because they are recognized as the cellular

elements responsible for the brain information processing [122]. In more practical terms, RNNs are non-linear statistical data modelling tools. RNNs are particularly suitable to solve problem in such applications like time series prediction, pattern and sequence recognition/classification, etc.

In this work, we develop an enzyme classification system based on a hybrid HMM-RNN gradient descent algorithm using real time recurrent learning of recurrent neural networks. We shall call the system as HMM-RNN-EC which stands for Hidden Markov Model - Recurrent Neural Network for Enzyme Classification. Some testings were undertaken to determine the training and generalization performance of the system.

Once the encoding scheme is defined, the next step is the construction of the RNN system for classification. In this work, we consider that there can be  $n$  different classes (numbers of families of enzymes), and in each class there can be  $m$  cases (number of enzymes). The length of a given sequence  $S_{ij}$  is defined as  $S_{ij}$  ( $i = 1, \dots, n; j = 1, \dots, m$ ). For the sake of simplification, we considered that all classes in the training set have the same  $m$ . But, in the testing set this was not necessary.

The hybrid algorithm can be applied into the following fields of bioinformatics: DNA sequencing, Sequence alignment, Sequence analysis, Protein structure, family prediction and so on.

Initially, we work on a sequence similarity problem of bio-informatics application. In particular, we aimed to find a closely related pattern by giving a particular sequence in a database. We have also explored the HMMER3 and PFAM softwares that identifies the pattern for a particular given sequence in its own generalized in-built databases, for example, the sequences in Humans, Rats, Plants, etc. We also compared the performances of HMMER3, PFAM and Hybrid HMM-RNN.

To search the sequences, there are two methods. The first method is to find the closely related patterns in Map20 Sequence file, i.e., we give one total sequence to find in map20 data set which consists of six sequences. The second method involves the use of the derived hybrid HMM-RNN algorithm. We search for closely related given pattern in a particular sequence training set.

We found the following results using PFAM HMMER:

Using the sequence

MTKVEPLKERAHDKTKAATTKNITKAPAKENKKPLEFKLHSGERAVKRAMFNYSVATNYY  
 IQKLQKKQEERLQKMIEEEEIRMLRKEMVPKAQLMPFFDRPFLPQRSSRPLTMPKEPSFG  
 NVNSTCWTCTVFNNQHLYHINHAHA

in *Pfam* and looking for protein search, we identified the following patterns:

query/39-95 LHSGERAVKRAMFNYSVATNYYIQKLQKKQEERLQK-MIEEEEEIRMLRKEM--VPKAQLM  
 Q9SJ62\_ARATH/248-304 FRLEERAERKKEFYMKLEEKIHAKKEVEKTNLQAKSK-ESQEEIIRLRKSL--TFKAGPM  
 Q94C48\_ARATH/209-265 FKCDQRAEKKEFYVKLEEKTHAKKEEINSMQAKSK-ETQEAELRMLRKSL--NFKATPM  
 Q9SSK3\_ARATH/269-325 FKCSERAERKKEFYMKLEEKIHAKKTETNQVQAKTQ-QKAEAEIKQFRKSL--NFKATPM  
 Q8LES4\_ARATH/167-223 FSSTSRLERRREFYQKLEEKQKALEAEKRENEKRLK-EEQEAVTKQLRKNM--AYKANPV  
 Q9LS82\_ARATH/232-288 FRSTERAERKKEFYTKLEEKHQAMEAEKTQSEARNK-EATEAALRQLRKSL--RFKANPM  
 Q8LAB8\_ARATH/129-185 FRSAQRAEKREYQKLEEKQALEAERNELEQRQK-DEQEAALKQLRKNL--KFKAKPV  
 Q9LU87\_ARATH/427-485 FRSDERAERKKEFFKVEEKNKKEKEDKFCGFKAN-QNTNLASEEHKNPQVGGFQVTPM  
 Q9I8N0\_XENLA/630-686 LATAKRAKERQEFDKCLAETEAQKSLLEEEIRKRR-EEEEKEEISQLRQEL--VHKAKPI  
 Q8BTJ3\_MOUSE/661-717 LATERRAKERQELEKMAEVEAWKLQQLLEVRQEE-EQKKEELARLRKEL--VHKANPI  
 Q9LF31\_ARATH/386-439 LHSDVRAVERAEFDYQVAEKMSFIEQYKMERERQK----EEEIRLRKEL--VPKAQPM  
 Q9FKW1\_ARATH/210-266 LHVDRPIERADFDHKIKEKEMMYKRHLEEAEEAAKM-VEEERALKQLRRTI--VPQTRPV.

Using our proposed hybrid HMM-RNN architecture, we obtained some results. We briefly describe the outcomes as follows:

The hybrid HMM-RNN architecture has been trained to test and predict the closely related sequences of a given pattern in a particular sequences of the training set. The results were reasonably fine. Then we worked on enzyme classification. The performance of the classified hybrid HMM-RNN-EC is very impressive and the following study reveals that any stochastic model based on HMM can be generalized and improve the accuracy or performance of the existing system.

### 5.3 Enzyme classification

In this section, we described about the actual real world problem implementation and its results. we studied the enzyme classification extensively in order to test an hybrid HMM-RNN system.

A number of experiments were done to test the performance of the Hybrid HMM-RNN system in the protein classification problem. The following issues were investigated: the

classification performance of the proposed system using a large dataset (of enzymes); the influence of the size of the training set in the learning process of the RNNs; the number of RNNs necessary for a satisfactory balance between computational time and classification performance.

Table 5.3.1: Kyte and Doolittle hydrophobicity scale

Amino Acid		K and D Scale	Type
Name	Symbol		
Isoleucine	I	4.5	Hydrophobic
Valine	V	4.2	Hydrophobic
Leucine	L	3.8	Hydrophobic
Phenylalanine	F	2.8	Hydrophobic
Cysteine	C	2.5	Hydrophobic
Methionine	M	1.9	Hydrophobic
Alanine	A	1.8	Hydrophobic
Glycine	G	-0.4	Natural
Threonine	T	-0.7	Natural
Serine	S	-0.8	Natural
Tryptophan	W	-0.9	Natural
Tyrosine	Y	-1.3	Natural
Proline	P	-1.6	Natural
Histidine	H	-3.2	Hydrophilic
Glutamine	Q	-3.5	Hydrophilic
Asparagine	N	-3.5	Hydrophilic
Glumatic acid	E	-3.5	Hydrophilic
Aspartic acid	D	-3.5	Hydrophilic
Lysine	K	-3.9	Hydrophilic
Arginine	R	-4.0	Hydrophilic

### Classification performance:

For this experiment, we used a whole enzyme superfamily extracted from PDB. A total of 3200 enzymes were used, divided into six families. The tables 5.3.1 and 5.3.2 summarizes the data used for the training set and the testing set.

All the results reported were obtained by performing a modified five-fold cross-validation procedure [85]. First, a given number of proteins were randomly drawn from the dataset for each of the six families. The sum of these samples constituted the training set (1200). All the

Table 5.3.2: HMM-RNN-EC - Training and testing set percentages

Class	Family	Training Set	Testing Set	Percentage(%)	Total
1	Oxidoreductases	200	354	17.7	554
2	Transferases	200	422	21.1	622
3	Hydrolases	200	918	45.9	1118
4	Lyases	200	164	8.1	364
5	Isomerases	200	92	4.6	292
6	Ligases	200	50	2.6	250
		1200	2000	100	3200

other proteins were allocated to the testing set (2000). Then, the neural system was trained and later evaluated using this partition. The accuracy rate on the testing set was computed as the ratio of the number of correctly classified proteins to the total number of proteins, as is standard in the literature. Next, a new sampling was taken from the dataset to form another training and testing set, and the training and testing processes were repeated. This procedure was repeated five times and the final results were reported as the averaged accuracy rate over these five runs.

The performance of our proposed hybrid HMM-RNN was compared with a classification procedure based on HMMs. For this purpose, we used the software package HMMER 3.0 (<http://hmmer.wustl.edu/>) that uses HMMs [57]. The HMMs are a well-known statistical modelling technique frequently applied to the analysis of time series and biological sequences [61]. The use of HMMER for protein classification encompasses three steps. First, a multiple sequence alignment is done using the training set. For this purpose we used the software *ClustalX, version 2.1*, for generating six files (one for each class) with the multiple alignment of the proteins. The second step is building a HMM that represents each class of the training set. This was done using as input the pre-aligned file mentioned before and the module HMMBUILD. This program creates a profile HMM for the family based on the examples given. Additionally, we used HMMCALIBRATE to optimize the generated models, so as to improve the classification performance. Finally, the third step is the classification of the testing set using the profile models generated. This was done using HMMPFAM.

The use of accuracy rate to assess classification performance is standard in the classification literature [85], but sometimes this measure can be misleading since it does not discrimi-



nate between positive and negative cases. That is, the accuracy rate is the sum of the correctly classified cases. Another useful way to measure a system classification performance is using sensitivity and specificity, two indicators commonly used in medical and life sciences. These measures are frequently used in two-class problems, but can be readily adapted for multiclass problems, as will be shown. When using a system for classifying a protein of unknown class, depending on the class predicted by the system and on the actual class of the protein, one of the following four types of result can be observed:

- True positive (TP) : The system predicts that the protein belongs to a given class and the protein really does belong to that class.
- False positive (FP) : The system predicts that the protein belongs to a given class but, in fact, it does not belong to it.
- True negative (TN) : The system predicts that the protein does not belong to a given class, and indeed it does not belong to it.
- False negative (FN) : The system predicts that the protein does not belong to a given class but, in fact, the protein does belong to it.

Based on these parameters, sensitivity (Se) and the specificity (Sp) can be defined as follows:

- $Se = TP / (TP + FN)$ ,
- $Sp = TN / (TN + FP)$ .

Sometimes sensitivity and specificity are called true positive rate and true negative rate, respectively. Sensitivity measures the ability of the classifier system to correctly assign a protein to its real class. In the other hand, specificity measures the ability of the system to reject a given protein as belonging to a class to which it does not belong.

For both approaches, specificity values were always higher than sensitivity. This means that both systems are more efficient at predicting when a given protein does not belong to a class than the opposite. Again, the Hybrid HMM-RNN system performed better than HMMER, when the values for both sensitivity and specificity are taken into account.

In the next section, we present the implementation details of enzyme classification using the hybrid HMM-RNN architecture.

## 5.4 Results and discussion

This section deals with the implementation results on enzyme classification using the hybrid HMM-RNN architecture.

In this study, our dataset consists of 6 enzymes superfamilies extracted from PDB. Here we used 3200 enzymes in total as the training and testing samples. The number of these samples constituted the training set is 1200. Other proteins were allocated to the testing set (2000). Table 5.3.2 summarized the data used for the training and the testing processes. We used a same number of enzymes in training for each class while for testing; number of samples for each class was according to percentage of total enzymes in PDB database, respectively. Total system have the same topology; one input layer with 40 input nodes, one hidden layer with 3, 5, 9, 10 and 15 hidden nodes and one output layer with 6 output nodes where each node represents one class of enzymes. The exception is in determining the number of hidden nodes, in which we applied two different approaches, i.e., using those proposed by [14, 188]. Table 5.4.1 presents the distributions of the training patterns and the number of hidden nodes in each of the network used in HMM-RNN-EC.

Table 5.4.1: Hybrid HMM-RNN for enzyme classification

Input Neurons	Training Epochs	Output Nodes	Training Set Size	Testing Set Size
40	300	6	1200	2000

To evaluate performance for our network system, we measured the accuracy in simulation for the testing set. Accuracy is defined as follows:

$$\text{Accuracy} = \frac{\text{Revealed Sequences}}{\text{Total number of Sequences Involved}} * 100. \quad (5.4.1)$$

Accuracy of the testing set is presented in the Table 5.4.2.

We adopted a new approach in encoding primary structure into real-valued vector that feeds into recurrent neural network using Kyte and Doolittle hydrophobicity scale [111]. Additionally, we used a set of networks with hybrid HMM-RNN algorithm in the weighting system. The results achieved showed that combination between HMM and RNN algorithm can be employed to optimize network structure. By using the method proposed by [188] in

Table 5.4.2: Accuracy of the testing set

Superfamily	Accuracy - 1	Accuracy - 2	Accuracy - 3	Accuracy - 4	Accuracy - 5
Oxidoreductases	78.11	77.29	75.33	78.29	79.33
Transferases	77.18	77.11	76.65	77.32	78.23
Hydrolases	74.64	72.20	71.18	79.13	78.46
Lyases	76.78	76.15	75.77	72.27	77.12
Isomerases	72.10	71.54	69.81	77.33	79.35
Ligases	61.45	60.11	58.86	63.54	67.11
Average	75.33	71.89	65.79	75.16	75.89

finding the optimal number of hidden nodes in the network, we successfully obtained good precision performance values for the testing datasets.

The accuracy rate for testing dataset with 6 classes of enzymes is reasonable (75.89 percent on average).

## 5.5 Summary

In this chapter, we have discussed the implementation of developed hybrid HMM-RNN architecture on a real world application such as Enzyme Classification. After having discussed the latest works in the field of bioinformatics, genetics and proteomics related to the hybrid systems of HMMs and RNNs, we have elaborated on some fundamental structure of molecular biology. As a first step, profile HMMs are implemented using the proposed hybrid HMM-RNN followed by results obtained using this approach. Then we implemented it on classifying the given sequence in a group of enzymes. Finally, we presented some results and compared them with those obtained by other existing methods.

In the next chapter, we present the concluding remarks and scope for further research.

# Chapter 6

## Concluding remarks and scope for further research

In this thesis, we have developed and implemented a HMM-RNN hybrid architecture using the gradient descent learning algorithm of real time recurrent neural networks on sample test beds and as well as on real time application. The trained hidden Markov model data is used as a prior knowledge into the recurrent neural network to process the knowledge stored in the networks. This is achieved by taking the advantages and disadvantages of both the models and through building a successful architecture. This allows for implementation of any generalized stochastic model on this network. We have then explored this approach for some biological applications.

The main contributions of the thesis are

- Design of a generalized hybrid HMM-RNN architecture for optimizing any stochastic model based on Hidden Markov Models.
- A mathematical derivation of a hybrid HMM-RNN architecture.
- Development of the hybrid HMM-RNN gradient descent learning algorithm using real time recurrent learning of recurrent neural networks.
- Implementation of the learning algorithm on sample test beds. It is achieved by modelling the dynamical systems such as deterministic finite state automata.

- Real world applications of the hybrid HMM-RNN algorithm to bioinformatics. Our results show that we can successfully classify a given protein sequence from the six different types of enzymes.

As far as the **scope for further research** is concerned, we indicate the following:

- Application of hybrid HMM-RNN architecture can be further explored to other applications (speech recognition, biometrics, etc.).
- Learning and representing the fuzzy finite state automata with the hybrid HMM-RNN can be worked further.
- The hybrid HMM-RNN architecture can be further explored in conjunction with the long short term memories of RNN and decoupled extended Kalman filters.



# Bibliography

- [1] Y.S. Abu-Mostafa, Learning from hints in neural networks, *Journal of Complexity*, 6:192, 1990.
- [2] E. Alba and J.F. Chicano, Training neural networks with GA hybrid algorithms, *Lecture Notes in Computer Science, Springer Berlin/Heidelberg*, 3102:852-863, 2004.
- [3] A.F. Ali and D.M. Shawky, A Novel Approach for Protein Classification Using Fourier Transform, *World Academy of Science, Engineering and Technology*, 44, 2010.
- [4] L.B. Almeida, A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment, *Proceedings of IEEE First Annual International Conference on Neural Networks*, 2:199, 1987.
- [5] R. Alquezar and A. Sanfeliu, An algebraic framework to represent finite state automata in single-layer recurrent neural networks, *Neural Computation*, 7(5):931-949, 1995.
- [6] R. Andrews, J. Diederich and A. Tickle, A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373-389, 1995.
- [7] P.J. Angeline, G.M. Saunders and J.B. Pollack, An evolutionary algorithm that builds recurrent neural networks, *IEEE Transactions on Neural Networks*, 5:54-65, 1994.
- [8] K. Asai, S. Hayamizu and H. Handa, Prediction of protein secondary structures by hidden marko models, *Computer Application in the Biosciences*, 9(2):141-146, 1993.
- [9] T.K. Attwood, M.E. Beck, A.J. Bleasby, K. Degtyarenko and D.J.P. Smith, Progress with the PRINTS protein fingerprint database. *Nucleic Acids Research*, 24:182-8, 1996.

- [10] P. Baldi, Y. Chauvin, T. Hunkapiller and M.A. McClure, Hidden Markov Models in Molecular Biology: New Algorithms and Applications, *In Advances in Neural Information Processing Systems*, 747-754, 1993.
- [11] P. Baldi, Y. Chauvin, T. Hunkapiller and M.A. McClure, Hidden Markov Models of Biological Primary Sequence Information, *Proceedings of the National Academy of Science*, 91:1059-1063, 1994.
- [12] P. Baldi, S. Brunak, P. Frasconi, G. Pollastri and G. Soda, Bidirectional dynamics for protein secondary structure prediction, *Lecture Notes in Computer Science*, 80-104:1828, 2001.
- [13] E. Barnard and D. Casasent, Invariance and neural networks, *IEEE Transactions on Neural Networks*, 2:498-508, 1991.
- [14] E.B. Baum and D. Haussler, What size net gives valid generalization?, *Neural Computation*. 1:151-160, 1989.
- [15] R. Bellman, *Adaptive control processes: A guided tour*, Princeton University Press, New Jersey, 1961.
- [16] K. Ben and V.D. Patrick, *An Introduction to Neural Network*, The University of Amsterdam, 1996.
- [17] Y. Bengio, P. Frasconi and P. Simard, learning long-term dependencies with recurrent neural networks, *IEEE International Conference on Neural Networks*, 3:157-166, 1993.
- [18] Y. Bengio, P. Simard and P. Frasconi, Learning long-term dependencies with gradient descent is difficult, *IEEE Transactions on Neural Networks*, 5(2):157-166, 1994.
- [19] Y. Bengio and P. Frasconi, An EM approach to learning sequential behavior, Technical Report RT-DSI-11/94, Universita di Firenze, 1994.
- [20] Y. Bengio, Neural Networks for Speech and Sequence Recognition, *International Thompson Computer Press*, 1996.

- [21] P.N. Bidargaddi, M. Chetty and J. Kamruzzaman, Combining segmental semi-Markov models with neural networks for protein secondary structure prediction, *Neurocomputing*, 72:3943-3950, 2009.
- [22] H. Bin Ma, Decentralized adaptive synchronization of a stochastic discrete-time multiagent dynamic model, *SIAM Journal on Control and Optimization*, 48(2):859-880, 2009.
- [23] E. Birney, Hidden Markov models in biological sequence analysis, *Deep computing for the life sciences*, 45, 2001.
- [24] C.M. Bishop, Neural Networks for Pattern Recognition, *Oxford University Press*, 1995.
- [25] A. Blanco, M. Delgado and M.C. Pegalajar, A real-coded genetic algorithm for training recurrent neural networks, *Neural Networks*, 14(1):93-105, 2001.
- [26] K. Blekas, D.I. Fotiadis and A. Likas, Motif-based protein sequence classification using neural networks, *Journal of Computational Biology*, 12:64-82, 2005.
- [27] C.L. Borro, R.M. Stanley Oliveira, E.B. Michel Yamagishi, L. Adauto Mancini, G. Jose Jardine, M. Ivan, H. Edgard dos Santos, H. Roberto Higa, R. Paula Kuser and G. Neshich, Predicting enzyme class from protein structure using Bayesian classification, *Genetic. Molecular Research*, 5(1): 193-202, 2006.
- [28] H. Bourlard and S. Bengio, Hidden Markov Models and other Finite State Automata, *The Handbook of Brain Theory and Neural Networks*, Second edition, (M.A. Arbib, Ed.), Cambridge, MA: The MIT Press, 2002.
- [29] S.J. Bridle, Alpha-nets: A recurrent neural network architecture with a hidden Markov model interpretation, *Speech Communication*, 9(1):83-92, 1990.
- [30] L. Bum, H. Lee and H. Keun, Design of a Novel Protein Feature and Enzyme Function Classification, *Computer and Information Technology Workshops*, 450-455, 2011.
- [31] O. Buse and A. Mutlu, Generalized classifier neural network, *Neural Networks*, 39:18-26, 2013.



- [32] C.Z. Cai, L.Y. Han, Z.L. Ji and Y.Z. Chen, Enzyme Family Classification by Support Vector Machines, *PROTEINS: Structure, Function, and Bioinformatics*, 55:66-76, 2004.
- [33] B. Cannas, S. Cincotti, A. Fanni, M. Marchesi, F. Pilo and M. Usai, Performance analysis of locally recurrent neural networks, *The International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, 17(6):708-716, 1998.
- [34] S. Carola, C. Antje and D. Schomburg, Development of a classification scheme for disease-related enzyme information, *BMC Bioinformatics*, 12:329, 2011.
- [35] R.C. Carrasco, M.L. Forcada, M.A Valdes and R.P. Neco, Stable encoding of finite-state machines in discrete-time recurrent neural nets with sigmoid units, *Neural Computation*, 12(9):2129-2174, 2000.
- [36] A. Ceroni, P. Frasconi and G. Pollastri, Learning protein secondary structure from sequential and relational data, *Neural Networks*, 18(8):1029-1039, 2005.
- [37] K.S. Chalup and D.A. Blair, Incremental training of first order recurrent neural networks to predict a context-sensitive language, *Neural Networks*, 16(7):955-972, 2003.
- [38] W. Cheng, J. Huang and C. Liou, Segmentation of DNA using simple recurrent neural network, *Knowledge-Based Systems*, 26:271-280, 2012.
- [39] V. Cherkassky. From statistics to neural networks: Theory and pattern recognition applications. In V. Cherkassky, J.H. Friedman, and H. Wechsler (eds.), *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, *NATO ASI Series F: Computer and System Sciences*, 136:127-146, 1994.
- [40] K. Chetan and A. Choudhary, A top-down approach to classify enzyme functional classes and sub-classes using random forest, *Journal on Bioinformatics and Systems Biology*, 2012. DOI:10.1186/1687-4153-2012-1.
- [41] S. Chiba, K. Sugawara and T. Watanabe, Classification and function estimation of protein by using data compression and genetic algorithms. *Proceedings of the 2001 Congress on Evolutionary Computation*, IEEE Press, 2:839-44, 2001.

- [42] T.J. Cholewo and J.M. Zurada. Sequential network construction for time series prediction. *Proceedings of the IEEE International Conference on Neural Networks*, IEEE Press, 2034-2039, 1997.
- [43] M.H. Christiansen and N.Chater, Toward a connectionist model of recursion in human linguistic performance, *Cognitive Science*, 23:417-437, 1999.
- [44] F. Christos and S. Andreas, Self-Organizing Hidden Markov Model Map (SOHMMM), *Neural Networks*, 48:133-147, 2013.
- [45] A. Cleeremans, D. Servan-Schreiber and J. McClelland, Finite-state automata and simple recurrent neural networks, *Neural Computation*, 1(3):372-381, 1989.
- [46] S. Das and R. Das, Induction of discrete state-machine by stabilizing a continuous recurrent neural network using clustering, *Journal of Computer Science and Informatics*, 21(2):35-40, 1991.
- [47] S. Das, C.L. Giles, and G.Z. Sun, Using hints to successfully learn context-free grammars with a neural network pushdown automaton. In S.J. Hanson, J.D. Cowan and C.L. Giles (eds.), *Advances in Neural Information Processing Systems 5*, San Mateo, CA, Morgan Kaufmann Publishers, 65-72, 1993.
- [48] K. David, D. Haussler, G. Martin Reese and H. Frank Eeckman, A generalized hidden Markov model for the recognition of human genes in DNA, *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, AAAI Press, 1996.
- [49] M.O. Dayoff, R.M. Schwartz and B.B. Orcutt, A Model of evolutionary change in proteins, Atlas of protein sequence and structure, *National Biomedical Resource Found*, 5:345-358, 1978.
- [50] L. Derong, X. Xiaoxu, Z. Hou and B. DasGupta, Identification of motifs with insertions and deletions in protein sequences using self-organizing neural networks, *Neural Networks*, 18(5-6):835-842, 2005.

- [51] R.T. De Souza, C.F. Caldas, R.S. Maria, S.C. Oliveira and A.P. Braga, Protein Classification with Extended-Sequence Coding by Sliding Window, *TCBB*, 8(6):1721-1726, 2011.
- [52] M. Des Jardins, P.D. Karp, M. Krummenacker, T.J. Lee and C.A. Ouzounis, Prediction of enzyme classification from protein sequence without the use of sequence similarity, *Proceedings of the International Conference on Intelligent System, Molecular Biology*, 5:92-9, 1997.
- [53] H. Diego Milone, E. Leandro Di Persia and E. Mar Torres, Denoising and recognition using hidden Markov models with observation distributions modeled by hidden Markov trees, *Pattern Recognition*, 43(4):1577-1589, 2010.
- [54] X. Ding and S. Canu, Neural Network Based Model for Forecasting, p153-165, *Neural Networks and their Applications*, edited by J. G. Taylor, *John Wiley and Sons*, 1996.
- [55] R.O. Duda, P.E. Hart and D.G. Stork, Pattern classification, *Wiley Publications*, 2001.
- [56] Md. Eamin Rahman, R. Islam, S.Islam, M. Shakhinur Islam and Md. Ruhul Amin, MiRANN: A reliable approach for improved classification of precursor microRNA using Artificial Neural Network model, *Genomics*, 99(4):189-194, 2012.
- [57] S. Eddy, Profile hidden Markov models, *Bioinformatics*, 14:755-763, 1998.
- [58] J.L. Elman and D. Zipser, Learning the hidden structure of speech, *Journal of the Acoustical Society of America*, 83:1615-1626, 1988.
- [59] J.L. Elman, Finding structure in time, *Cognitive Science*, 14:179-211, 1990.
- [60] J.L. Elman, Distributed representations, simple recurrent networks, and grammatical structure, *Machine Learning*, 7:195-226, 1991.
- [61] E. Eskin, W.S. Noble and Y. Singer, 2003. Protein family classification using sparse Markov transducers, *Journal of Computational Biology*, 10:187-214, 2003.
- [62] I. Evangelia, G. Pantelis, I. Zoi and J. Stavros Hamodrakas, PredSL: A Tool for the N-terminal Sequence-based Prediction of Protein Subcellular Localization, *Genomics, Proteomics and Bioinformatics*, 4(1):48-55, 2006.

- [63] L. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall, 1994.
- [64] L. Feldkamp, D. Prokhorov, C. Eagen and F. Yuan, Enhanced multi-stream kalman filter training for recurrent networks, *Nonlinear Modeling: Advanced Black-Box Techniques*, 29-53, 1998.
- [65] M. L. Forcada and R. C. Carrasco, Learning the initial state of a second-order recurrent neural network during regular-language inference, *Neural Computation*, 7(5):923-930, 1995.
- [66] P. Frasconi, M. Gori, M. Maggini and G. Soda. Unified integration of explicit rules and learning by example in recurrent networks, *IEEE Transactions on Knowledge and Data Engineering*, 7(2):340-346, 1995.
- [67] L.M. Fu and L.C. Fu, Mapping rule-based systems into neural architecture, *Knowledge-Based Systems*, 3(1):48-56, 1990.
- [68] L.M. Fu, Knowledge-based connectionism for revising domain theories, *IEEE Transactions on Systems, Man and Cybernetics*, 23(1):173-182, 1993.
- [69] L. Fu, Rule generation from neural networks, *IEEE Transactions on Systems, Man and Cybernetics*, 24(8):1114-1124, 1994.
- [70] Y. Fukuoka and H. Matsuki, A Modified Back-propagation method to avoid local minima, *Neural Networks*, 11:1059-1072, 1998.
- [71] M.J.F. Gales, Maximum likelihood linear transformations for Hidden Markov Model-based speech recognition, *Computer Speech Language*, 12:75-98, 1998.
- [72] S. Geman, E. Bienenstock and R. Dourstat, Neural networks and the bias/variance dilemma, *Neural Computation*, 4(1):1-58, 1992.
- [73] G.R. George and F. Cardullo, Application of Neuro-Fuzzy Systems to Behavioral Representation in Computer Generated Forces, *Proc. of the Eighth Conference on Computer Generated Forces and Behavioral Representation*, 575-585, 1999.

- [74] F.A. Gers, J. Schmidhuber and F.A. Cummins, Learning to forget: Continual prediction with LSTM, *Neural Computation*, 12(10):2451-2471, 2000.
- [75] F. Gers and J. Schmidhuber, LSTM networks learn simple context free and context sensitive languages, *Technical Report*, IDSEA, Manno, Switzerland, 2001.
- [76] F. Gers, D. Eck and J. Schmidhuber, Applying LSM to time series prediction through time-window approaches, *Technical Report*, IDSEA, Manno, Switzerland, 2002.
- [77] F. Gers, N. Schraudolph and J. Schmidhuber, Learning precise timing with LSTM recurrent networks, *Journal of Machine Learning Research*, 3:115-143, 2002.
- [78] L.S. Goh and P.D. Mandic, An augmented CRTRL for complex-valued recurrent neural networks, *Neural Networks*, 20(10):1061-1066, 2007.
- [79] M. Golea, On the complexity of rule-extraction from neural networks and network-querying. Technical report, Department of Systems Engineering, Australian National University, 1996.
- [80] M.W. Goudreau, C.L. Giles, S.T. Chakradhar and D.Chen, First-order vs second-order single layer recurrent neural networks, *IEEE Transactions on Neural Networks*, 5(3):511-513, 1994.
- [81] A. Graves, Supervised Sequence Labelling, *Studies in Computational Intelligence Springer*, 385:1-131, 2012.
- [82] A. Guillermin, A. Gustavo de la Riva, R. Molina-Ruiz, A. Sanchez-Rodriguez, P. Giselle, V. Vitor and A. Antunes, Non-linear models based on simple topological indices to identify RNase III protein members, *Journal of Theoretical Biology*, 273(1):167-178, 2011.
- [83] R. Hadsell, P. Sermanet, E. Ayse, J. Ben, J. Han, F. Beat, U. Muller and Y. LeCun, Online Learning for Offroad Robots: Using Spatial Label Propagation to Learn Long-Range traversability, *Proceedings of Robotics Science and Systems Conference*, 2007.

- [84] Z. Haiquan, Z. Xiangping, J. Zhang, Y. Liu, X. Wang and T. Li, A novel joint-processing adaptive nonlinear equalizer using a modular recurrent neural network for chaotic communication systems, *Neural Networks*, 24(1):12-18, 2011.
- [85] D.J. Hand, Construction and assessment of classification rules, *John Wiley & Sons*, New York, USA, 1997.
- [86] Y. Hayashi and A. Imura, Fuzzy neural expert systems with automated extraction of fuzzy if-then rules from a trained neural network, *Proceedings of First IEEE Conference on Fuzzy Systems*, 489-494, 1990.
- [87] J. Henderson, S. Salzberg and K. Fasman, Finding genes in human DNA with a hidden Markov model, *Journal of Computational Biology*, 4:127-141, 1997.
- [88] J. Henriques and A. Dourado, A Multivariable adaptive control using a recurrent neural network, *Proceedings of the Engineering Applications of Neural Networks*, 9(12):118-121, 1998.
- [89] J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, 1991.
- [90] O. Hiroyuki and G. Yukio Pegio, Recurrent neural network architecture with pre-synaptic inhibition for incremental learning, *Neural Networks*, 19(8):1106-1119, 2006.
- [91] S. Hochreiter, Y. Bengio, P. Frasconi and J. Schmidhuber, Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, *Field Guide to Dynamic Recurrent Networks*, 237-243, 2001.
- [92] S. Hochreiter and J. Schmidhuber, Long Short Term memory, *Neural Computation*, 9(8):1735-1780, 1997.
- [93] J.H. Holland, *Adaption in natural and artificial systems*, University of Michigan Press, Cambridge, 1975.
- [94] J.J. Hopfield, Neural networks and physical systems with emergent collective computational facilities, *Proceedings of the National Academy of Sciences of the USA*, 79:2554-2558, 1982.

- [95] B. Horne and C. Giles, An experimental comparison of recurrent neural networks, *Advances in Neural Information Processing Systems*, 7:697-704, 1995.
- [96] K. Hornik, M. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, 2:359-366, 1989.
- [97] A. Hosseini, J. Wang and S. Mohammad, A recurrent neural network for solving a class of generalized convex optimization problems, *Neural Networks*, 44:78-86, 2013.
- [98] Y. Hu, Biopattern discovery by genetic programming, In Koza Jr. (ed.) *Proceedings of the Third Annual Genetic Programming Conference*, Madison, USA, San Francisco: Morgan Kaufmann Publishers, 152-7, 1998.
- [99] R. Hughey and A. Krogh, Hidden Markov models for sequence analysis: extension and analysis of the basic method, *Computer Applications in the Biosciences*, 12:95-107, 1996.
- [100] S. Ilya and G. Hinton, Temporal-Kernel Recurrent Neural Networks, *Neural Networks*, 23(2):239-243, 2010.
- [101] G. Ivan and D. Andrej, On-line identification and reconstruction of finite automata with generalized recurrent neural networks, *Neural Networks*, 16(1):101-120, 2003.
- [102] R.A. Jacobs, M.I. Jordan, S.J. Nowlan and G.E. Hinton, Adaptive mixtures of local experts, *Neural Computation*, 3:79-87, 1991.
- [103] H. Jaeger, The echo state approach to analyzing and training recurrent neural networks, GMD Report 148, German National Research Center for Information Technology, 2001.
- [104] Q. Jian-Ding, L. San-Hua, H. Jian-Hua and L. Ru-Ping, Support vector machines to distinguish enzymes: Approached by incorporating wavelet transform, *Journal of Theoretical Biology*, 256(4-21):625-631, 2009.
- [105] M.I. Jordan, Attractor dynamics and parallelism in a connectionist sequential machine, In *Proceedings of the Ninth Annual conference of the Cognitive Science Society*, Lawrence Erlbaum, 531-546, 1986.

- [106] K.N. Kasabov, V. Jain and B. Lubica Lau, Integrating evolving brain-gene ontology and connectionist-based system for modeling and knowledge discovery, *Neural Networks*, 21(2-3):266-275, 2008.
- [107] J.F. Kolen and S.C. Kremer, (ed.), *A Field Guide to Dynamical Recurrent Networks*, IEEE Press, Piscataway, NJ, 2001.
- [108] J.R. Koza, Classifying protein segments as transmembrane domains using genetic programming and architecture-altering operations, In T. Back, D.B. Fogel, Z. Michalewicz, eds., *Handbook of Evolutionary Computation*, UK: Institute of Physics Publishing, 6:1-5, 1997.
- [109] A. Krogh, M. Brown, I.S. Mian, K. Sjolander and D. Haussler, Hidden Markov Models in Computational Biology: Applications to Protein Modeling, *Journal of Molecular Biology*, 235:1501-1531, 1994.
- [110] A. Krogh, In S.L. Salzberg *et al.*, An Introduction to Hidden Markov Models for Biological Sequences, *Computational Methods in Molecular Biology*, Elsevier, 45-63, 1998.
- [111] J. Kyte and R. Doolittle, A simple method for displaying the hydropathic character of proteins, *Journal of Molecular Biology*, 157:105-32, 1982.
- [112] S. Lawrence, C.L. Giles and S. Fong, Natural Language Grammatical Inference with Recurrent Neural Networks. *IEEE Transactions on Knowledge and Data Engineering*. 12(1):126-140, 2000.
- [113] S. Lawrence, S. Fong and C.L. Giles, Natural language grammatical inference: A comparison of recurrent neural networks and machine learning methods. In S. Wermter, E. Riloff, and G. Scheler (eds.), *Symbolic, Connectionist, and Statistical Approaches to Learning for Natural Language Processing*, Lecture notes in AI, Springer-Verlag, Berlin, 33-47, 1996.
- [114] S. Lawrence, C.L. Giles and A.C. Tsoi, Symbolic conversion, grammatical inference and rule extraction of foreign exchange rate prediction, *Decision Technologies for Financial Engineering: Proceedings of the Fourth International Conference on Neural Networks in the Capital Markets*, World Scientific, Singapore, 333-345, 1998.



- [115] Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard and L.D. Jackel, Backpropagation applied to handwritten zip code recognition, *Neural Computation*, 1:541-551, 1989.
- [116] A.L. Lehninger, D.L. Nelson and M.M. Cox, Principles of biochemistry with an extended discussion of oxygen-binding proteins, New York: Worth Publishers Inc., 1998.
- [117] T. Lin, B.G. Horne, P. Tino and C.L. Giles, Long-term dependencies in NARX Neural Networks, *World Congress in Neural Networks*, 3:142-146, 1995.
- [118] J.P. Lisboa and F.G.T. Azzam, The use of artificial neural networks in decision support in cancer: A systematic review, *Neural Networks*, 19(4):408-415, 2006.
- [119] P. Manolios and R. Fanelli, First order Recurrent Network and deterministic finite state automata, *Neural Computation*, 6(6):1154-1172, 1994.
- [120] K. Marakami and H Taguchi, Gesture recognition using recurrent neural networks, *Proceedings of the SIGCHI conference on Human factors in computing systems*, 237-242, 1991.
- [121] F. Marco, B. Alberto, R. Matteo and V. Giorgio, A neural network algorithm for semi-supervised node label learning from unbalanced data, *Neural Networks*, 43:84-98, 2013.
- [122] E.D. Martin and A. Araque, Astrocytes and the biological neural networks, *Artificial Neural Networks in Real Life Applications*, 22-45, 2006.
- [123] W.S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, 5:115-33, 1943.
- [124] C. McMillan, M.C. Mozer and P. Smolensky, Rule induction through integrated symbolic and subsymbolic processing, In J. Moody, S. Hanson, and L. Giles (eds.), *Advances in in Neural Information Processing Systems 4*, San Mateo, CA, Morgan Kaufmann, 1992.
- [125] F. Michael, S. Li, X. Fu, A. Eduardo and W. Donald, An adaptive recurrent neural-network controller using a stabilization matrix and predictive inputs to solve a tracking problem under disturbances, *Neural Networks*, 49:74-86, 2014.

- [126] H. Mike, N. Lydia, P. Damon, M. John, L. Chris, F. Sky, F. Vance and ed., Multi-scale correlation structure of gene expression in the brain, *Neural Networks*, 24(9):933-942, 2011.
- [127] M.L. Minsky, Logical vs. analogical or symbolic vs. connectionist or neat vs. scruffy, In Patrick H. Winston, editor, *Artificial Intelligence at MIT, Expanding Frontiers*, MIT Press, Cambridge, MA, 1, 1990.
- [128] A. Mohammed and C. Guda, Computational approaches for automated classification of enzyme sequences, *Journal of Proteomics and Bioinformatics*, 23(4):147-152, 2011.
- [129] R. Mooney, J. Shavlik, J. Towell and A. Gove, An experimental comparison of symbolic and connectionist learning programs, *Proceedings of the International Joint Conference on Artificial Intelligence*, 775-780, 1989.
- [130] M.C. Mozer, A focused backpropagation algorithm for temporal pattern processing, *Complex Systems*, 3(4):349-381, 1989.
- [131] M.C. Mozer and S. Das, A connectionist chunker that induces the structure of context-free languages, In S.J. Hanson, J.D. Cowan, and C.L. Giles (eds.), *Advances in Neural Information Processing Systems 5*, MorganKaufmann Publishers, San Mateo, CA, 1993.
- [132] M.C. Mozer, Neural net architectures for temporal sequence processing, In A.S. Weigend and N.A. Gershenfeld (eds.), *Time Series Prediction*, Addison-Wesley, 243-264, 1994.
- [133] A.G. Murzin, S.E. Brenner, T. Hubbard and C. Chothia, SCOP: A structural classification of proteins database for the investigation of sequences and structures, *Journal of Molecular Biology*, 247:536-40, 1995.
- [134] A. Narayanan, E.C. Keedwell and B. Olsson, Artificial intelligence techniques for bioinformatics, *Applied Bioinformatics*, 1(4):191-222, 2002.
- [135] N. Neetika, B. John and O. Mitchell, Classification of Enzymes via Machine Learning Approaches, 2011.
- [136] T. Ohkawa, D. Namihira, N. Komoda and H. Nakamura, Protein structure classification by structural transformation. In N.G. Bourbakis (ed), *Proceedings of IEEE International*

- Joint Symposia on Intelligence and Systems*, Rockville, USA, Los Alamitos, USA: IEEE Computer Society Press, 23-9, 1996.
- [137] H.M. Osman, L. Choong-Yeun and I. Hashim, Hybrid Learning Algorithm in Neural Network System for Enzyme Classification, *Int. J. Advance Soft Computing Applications*, 2(2), 2010.
- [138] B. Pandey and R.B. Mishra, Knowledge and intelligent computing system in medicine, *Computers in Biology and Medicine*, 39(3):215-230, 2009.
- [139] A.G. Parlos, O.T. Rais and A.F. Atiya, Multi-step-ahead prediction using dynamic recurrent neural networks, *Neural Networks*, 13(7):765-786, 2000.
- [140] M.J. Pazzani and D. Kibler, The utility of knowledge in inductive learning, *Machine Learning*, 2:57-94, 1992.
- [141] B.A. Pearlmutter, Earning state space trajectories in recurrent neural networks, *Neural Computation*, 1(2):263-269, 1989.
- [142] R. Perkins and A. Brabazon, Predicting credit ratings with a GA-MLP hybrid, in *Artificial Neural Networks in Real Life Applications*, J.R. Rabunal and J. Dorrado, eds., IGI Global, USA, 220-237, 2006.
- [143] V. Petridis and A. Kehagias, Predictive modular neural networks: Applications to Time Series, Kluwer Academic Publishers, 123-133, 1998.
- [144] J.B. Pollack, Recursive distributed representations, *Artificial Intelligence*, 46:77-105, 1990.
- [145] E.T. Portegys, A maze learning comparison of Elman, long short-term memory and Mona neural networks, Original Research Article, *Neural Networks*, 23(2):306-313, 2010.
- [146] D.V. Prokhorov, G.V. Puskorius and L.A. Feldman, Dynamical recurrent networks in control, In J.F. Kolen and S.C. Kremer (eds.), *A Field Guide to Dynamical Recurrent Networks*, IEEE, Piscataway, NJ, 257-289, 2001.

- [147] G.V. Puskorius and L.A. Feldkamp, Decoupled extended Kalman filter training of feed-forward layered networks, *Proceedings of the International Joint Conference on Neural Networks*, 1:771-777, 1991.
- [148] W. Quen, J. Chang and S. Lee, On-line signature verification using LPC cepstrum and neural networks, *IEEE Transactions on Systems, Man and Cybernetics*, 27(1):148-153, 1997.
- [149] L.R. Rabiner and B.H. Juang, An Introduction to Hidden Markov Models, *IEEE ASSP Magazine*, 1-16, 1986.
- [150] L.R. Rabiner, A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition , *Proceedings of the IEEE*, 77(2):257-285, 1989.
- [151] J.R. Rabunal and J. Puertas, Hybrid system with artificial neural networks and evolutionary computation in civil engineering, in *Artificial Neural Networks in Real Life Applications*, 166-187, 2006.
- [152] P. Razvan and J. Herbert, A neurodynamical model for working memory, *Neural Networks*, 24(2):199-207, 2011.
- [153] R.D. Reed and J. Robert Mark, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, The MIT Press, 1999.
- [154] B.D. Ripley, *Pattern Recognition and Neural Networks*, Cambridge University Press, 1996.
- [155] A.J. Robinson, An application of recurrent neural nets to phone probability estimation, *IEEE transactions on Neural Networks*, 5(2):298-305, 1994.
- [156] P. Rodriguez, Simple recurrent networks learn context-free and context-sensitive languages by counting, *Neural Computation*, 13(9), 2001.
- [157] P. Rodriguez, J. Wiles and J.L. Elman, A recurrent neural network that learns to count, *Connection Science*, 11:5-40, 1999.

- [158] X. Rui, K.G. Venayagamoorthy and C.D. Wunsch, Modeling of gene regulatory networks with hybrid differential evolution and particle swarm optimization, *Neural Networks*, 20(8):917-927, 2007.
- [159] D.E. Rumelhart, J.L. McClelland and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Foundations, MIT Press, Cambridge, MA, 1, 1986.
- [160] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning internal representations by error propagation, *Nature*, 323:533-536, 1986.
- [161] W.S. Sarles, Neural Network faq, periodic posting to the Usenet newsgroup, 1997.
- [162] J. Schmidhuber, Learning complex extended sequences using the principle of history compression, *Neural Computing*, 4(2):234-242, 1992.
- [163] U. Seiffert and M.H. Osman *et al.*, Multiple layer perceptron training using genetic algorithm, *Proceedings of European Symposium on Artificial Neural Network*, 12, 2001.
- [164] B. Sepideh, G. Amir and S. Ali Seyyedsalehi, Towards designing modular recurrent neural networks in learning protein secondary structures, *Expert Systems with Applications*, 39(6):6263-6274, 2012.
- [165] S.S. Seyhan, N.F. Alpaslan and Y. Mustafa, Simple and complex behavior learning using behavior hidden Markov model and CobART, *Neurocomputing*, 103:121-131, 2013.
- [166] V. Sharma and D. Srinivasan, A hybrid intelligent model based on recurrent neural networks and excitable dynamics for price prediction in deregulated electricity market, *Engineering Applications of Artificial Intelligence*, 26(5-6):1562-1574, 2013.
- [167] J.W. Shavlik, Combining symbolic and neural learning, *Machine Learning*, 14(3):321-331, 1994.
- [168] C. Shibata and R. Yoshinaka, A comparison of collapsed Bayesian methods for probabilistic finite automata, *Machine Learning*, 2013.
- [169] H. Siegelmann, B. Horne and C.L. Giles, Computational capabilities of recurrent narx neural networks, *IEEE Transactions on Systems, Man and Cybernetics*, 27(2):208, 1997.

- [170] J. Sima, Neural Expert Systems, *Neural Networks*, 2:261-271, 1995.
- [171] A.D. Smith, Oxford dictionary of biochemistry and molecular biology, Oxford University Press, Oxford, 1997.
- [172] A. Sperduti, On the Computational Power of Recurrent Neural Networks for Structures, *Neural Networks*, 10(3):395-400, 1997.
- [173] R. Sun and L. Bookman, eds. *Computational Architectures Integrating Neural and Symbolic Processes*, Kluwer Academic Publishers, 1994.
- [174] A. Tan and H. Pan, Predictive neural networks for gene expression data analysis, *Neural Networks*, 18(3):297-306, 2005.
- [175] N. Takehiko, Theoretical analysis of batch and on-line training for gradient descent learning in neural networks, *Neurocomputing*, 73(1-3):151-159, 2009.
- [176] J.G. Thistle and W.M. Wonham, Control of infinite behavior of finite automata, *SIAM Journal on Control and Optimization*, 32(4):1075-1097, 1994.
- [177] J.A. Torkestani, An adaptive focused Web crawling algorithm based on learning automata, *Applied Intelligence*, 37(4):586-601, 2012.
- [178] W. Tiffin, *Advanced Algorithms for Neural Networks: a C++ Sourcebook*. John Wiley and Sons, Inc., 1995.
- [179] G.G. Towell and J.W. Shavlik, Using symbolic learning to improve knowledge-based neural networks, *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA, 177-182, 1992.
- [180] G.G. Towell and J.W. Shavlik, The extraction of refined rules from knowledge-based neural networks, *Machine Learning*, 13(1):71-101, 1993.
- [181] G.G. Towell and J.W. Shavlik, Knowledge-based artificial neural networks, *Artificial Intelligence*, 70(1-2):119-165, 1994.
- [182] E. Trentin and R. Cattoni, A Hybrid Framework for Indoor Robot Navigation, *Sequence Learning, Perspectives in Neural Computing*, 255-263, 1999.

- [183] B. Trakhenbrot and Y. Barzdin, *Finite Automata: Behaviour and Synthesis*, North Holland, Amsterdam, 1973.
- [184] A.C. Tsoi and T. Back, Locally recurrent globally feedforward networks, a critical review of architectures, *IEEE Transactions on Neural Networks*, 5(2):229-239, 1994.
- [185] P.E. Utgoff, Incremental induction of decision trees, *Journal of Machine Learning*, 4:161-186, 1990.
- [186] K. Vered, S. Zach, E. Shimon, R. Eytan and H. David, Motif Extraction and Protein Classification, *IEEE Computational Systems Bioinformatics Conference (CSB'05)*, 0-7695-2344-7, 2005.
- [187] V. Viola, A. Alessandro and G. Pollastri, Accurate prediction of protein enzymatic class by N-to-1 Neural Networks, *BMC Bioinformatics*, 14(1):S11, 2013.
- [188] N. Wanas, G. Auda, M.S. Kamel and F. Karray, On the optimal number of hidden nodes in a neural network, *IEEE Canadian Conference on Electrical and Computer Engineering*, 1998.
- [189] J.T.L. Wang, Q. Ma, D. Shasha and C.H. Wu, New techniques for extracting feature from protein sequences, *IBM: System Journal*, 40:426-441, 2001.
- [190] J.T.L. Wang, Q. Ma, D. Shasha and C.H. Wu, Application of neural networks to biological data mining: a case study in protein sequence classification, In R. Ramakrishnan ed., *Proceedings of 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Boston, USA, New York, USA: ACM Press., 2003.
- [191] R.L. Watrous and G.M. Kuhn, Induction of finite-state automata using second-order recurrent networks, *Proceedings of Advances in Neural Information Systems*, California, USA, 309-316, 1992.
- [192] A.S. Weigend, D.E. Rumelhart and B.A. Huberman. Generalization by weight-elimination with application to forecasting, In R.P. Lippmann, J.E. Moody and D.S. Touretzky (eds.), *Advances in Neural Information Processing Systems, Proceedings of the 1990 Conference*, Morgan Kaufmann, San Mateo, CA, 875-882, 1991.

- [193] A.S. Weigend and N.A. Gershenfeld, The Future of Time Series, Learning and Understanding, *Addison-Wesley*, Reading, MA, 1-17, 1993.
- [194] R.W. Weinert and S.H. Lopes, A Neural Network System for Enzyme Classification, *Applied Bioinformatics*, 3(1):41-48, 2004.
- [195] A. Wu and Z. Zeng, Dynamic behaviors of memristor-based recurrent neural networks with time-varying delays, *Neural Networks*, 36:1-10, 2012.
- [196] W. Wu, J. Wang, M. Cheng and Z. Li, Convergence analysis of online gradient method for BP neural networks, *Neural Networks*, 24(1):91-98, 2011.
- [197] P.J. Werbos, Backpropagation through time; what it does and how to do it, *Proceedings of the IEEE*, 78:1550-1560, 1990.
- [198] R.J. Williams and D. Zipser, A learning algorithm for continually running fully recurrent neural networks, *Neural Computation*, 1(2):270-280, 1989.
- [199] R.J. Williams, Training recurrent networks using the extended Kalman filter, In *International Joint Conference on Neural Networks*, 4:241-250, 1992.
- [200] R.J. Williams and D. Zipser, Gradient-based learning algorithms for recurrent networks and their computational complexity, In: Y. Chauvin and D.E. Rumelhart (eds.), *Backpropagation: Theory, Architectures and Applications*, Hillsdale, NJ: Erlbaum, 433-486, 1995.
- [201] C.H. Wu, G.M. Whitson and G.J. Montllor, PROCANS: a protein classification system using a neural network, *Proceedings of IEEE International Joint Conference on Neural Networks*, IEEE Computer Society Press, 2:91-6, 1990.
- [202] H. Wu and F. Noe, Probability distance based compression of Hidden Markov Models, *Society for Industrial and Applied Mathematics*, 8(5):1838-1861, 2010.
- [203] D. Xeumei and D. Zhou, Learning gradients by a gradient descent algorithm Original Research Article, *Journal of Mathematical Analysis and Applications*, 341(2):1018-1027, 2008.



- [204] F. Zamora-Martinez, V. Frinken, S. Espana-Boquera, M.J. Castro-Bleda, A. Fischer and H. Bunke, Neural Network Language Models for Off-Line Handwriting Recognition, *Pattern Recognition*, In Press, 2013.
- [205] Y. Zhang and F. Cao, Analysis of convergence performance of neural networks ranking algorithm, *Neural Networks*, 34, 65-71, 2012.
- [206] X. Zhao, Y. Cheung and D. Huang, A novel approach to extracting features from motif content and protein composition for protein sequence classification, *Neural Networks*, 18(8):1019-1028, 2005.
- [207] P.V. Zhdanov, Three generic bistable scenarios of the interplay of voltage pulses and gene expression in neurons, *Neural Networks*, 44:51-63, 2013.
- [208] A. Zimek, F. Buchwald, E. Frank and S. Kramer, A Study of Hierarchical and Flat Classification of Proteins, *IEEE/ACM Trans Computational Biology, Bioinformatics*, 7(3):563-71, 2010.

